Laboratory 2 (Yr 06-07)

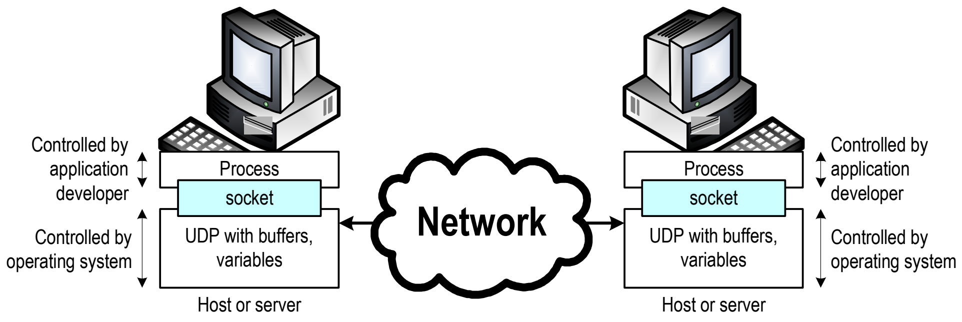
Socket programming in Java

Learning Objectives

The purpose of this laboratory is to teach you the basic skill of socket programming in Java. After completing this laboratory, you would be able to send and receive the signaling messages through the datagram sockets.

1. Introduction

Fig. 2.1 illustrates *socket communication* between two processes that communicate over a network. The two processes communicate with each other by sending and receiving messages through their *sockets*. A socket is the interface between the application layer and the transport layer within a host. The application developer has control of everything on the application-layer side of the socket, but has little control of the transport-layer side of the socket (e.g., the choice of transport protocol for the communication) [1].



Controlled by application developer
Controlled by operating system
Process
socket
UDP with buffers, variables
Host or server
Network
Process
socket
UDP with buffers, variables
Host or server
Controlled by application developer
Controlled by operating system

Fig. 2.1 Application processes, sockets, and the underlying transport protocol [1].

There are two transport protocols that can be used for socket programming in Java [2]:

Stream communication: It uses TCP (Transmission Control Protocol) for data transmission. This protocol guarantees reliable and in-order delivery of sender to receiver data. TCP support many popular applications on the Internet, including the World Wide Web, email, and Secure Shell. However it may not suitable for real-time communications.

Datagram communication: It uses UDP (User Datagram Protocol) for data transmission. UDP does not provide the reliability and ordering guarantees that TCP does, datagram may arrive out of order or go missing without notice. However, UDP is faster and more efficient for many lightweight or time-sensitive purposes.

Since we are going to build an Internet phone system, we therefore focus on socket programming using the UDP.

2. Processing Address

In socket communication, the sending process identifies the receiving process through two pieces of information: (1) the *IP address*, and (2) the *port number*. The IP address is a 32-bit quantity (e.g., 10.10.10.1) that uniquely identifies the interface by which the host is connected to the network. The port number is used to identity the receiving process on the destination host, so that the receiving end system can direct the message to that receiving process on that system [see Fig. 2.2].

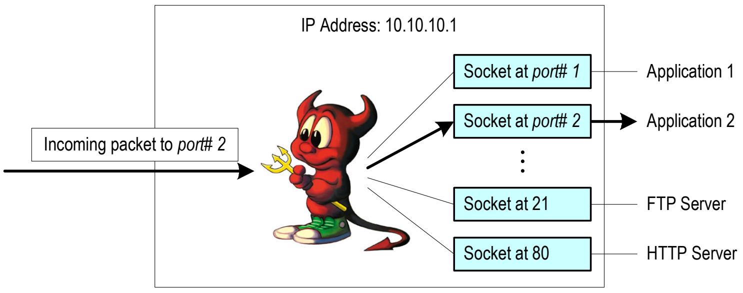


Fig. 2.2 Directing the incoming message to the appropriate application.

In Java, the IP address is represented by `InetAddress` class, which contains many methods for handling the IP address [see Appendix]. For example:

- **`static`** `InetAddress getLocalHost()` – It returns the local host.
- **`static`** `InetAddress getByName(String host)` – It determines the IP address of the given host name.
- `String getHostAddress()` – It returns the IP address in `String` for presentation.

Exercise 2.1

Write a program to display the IP address of you computer.

3. Socket Programming Using Datagram

In datagram communication, Java uses `DatagramPacket` and `DatagramSocket` to send and receive packets over the network. The two classes are described as follows.

`DatagramPacket`: It represents a datagram packet. It is used to implement a connectionless packet delivery service. Each message is routed from one machine to another, based solely on information contained within the packet.

`DatagramSocket`: It represents a socket for sending and receiving datagram packets. It is the sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed.

The programming steps for receiving data using datagram are as follows.

i) Create a `DatagramSocket` for listening incoming packet at a given port number (e.g., 2000).
e.g., `DatagramSocket socket = new DatagramSocket(2000);`

ii) Create a `byte` array to hold the received data.
e.g., `byte[] buf = new byte[256];`

iii) Create a `DatagramPacket` for holding the received packet.
e.g., `DatagramPacket packet = new DatagramPacket(buf, buf.length);`

iv) Wait for the packet.
e.g., `socket.receive(packet);`

v) Get data from the received packet.
e.g., `String message = new String(packet.getData()).trim();`

The programming steps for sending data using datagram are as follows.

i) Create a `DatagramSocket` from which the packet will send out.
e.g., `DatagramSocket socket = new DatagramSocket();`

ii) Create a byte array that holds the data (e.g., `String message = "END";`) to be sent.
e.g., `byte[] buf = message.getBytes();`

iii) Create a `DatagramPacket` with the IP address and the port number of the receiver.
e.g., `DatagramPacket packet = new DatagramPacket(buf, buf.length, targetIP, 2000);`

iv) Send the packet.
e.g., `socket.send(packet);`

v) Close the socket.
e.g., `socket.close();`

Exercise 2.2

1. Write a program with a class `Messenger` that repeatedly listens on a given port (e.g., 2000). Whenever a packet arrived, it will display the received message, IP address, and port number of the received packet (i.e., the port number on the remote host from which the datagram is being sent). If an "END" message is received, it will terminate the program.

```
/**
* File name: Messenger.java
*/
import java.net.*;

public class Messenger {

  int messagePort;

  public Messenger(int aPort) {
    messagePort = aPort;
  }

  public void start() {
    // Create DatagramSocket on the messagePort
    ...
    // Create byte array that hold the data to be sent
    byte[] buf = new byte[256];
    // Create DatagramPacket to hold the received data
    ...
    while (true) {
      // Empty the byte array
      for (int i=0; i<256; i++) buf[i] = (byte)0;
      System.out.println("Waiting for a packet at port# " + messagePort);
      // Wait for packet
      ...
      // Get message from the received packet
      ...
      // Print the message, IP address, and port number of the received packet
      ...
      // Check for termination
      ...
    }
  }
}
```

2. Modify the class `Messenger` by adding a `send` method. This `send` method can be used to send message to a given IP address and port number.

```
  public void send(InetAddress aTargetIp, int aTargetPort, String aMessage) {
    // Create DatagramSocket at any available port number
    ...
    // Create byte array to hold the received data
    ...
    // Create DatagramPacket
    ...
    // Send the packet
    ...
    // Close the socket
    ...
  }
```

3. Write a program to test the `Messenger` class.

   Hint: You can execute two instances of you program, one for listening and other for sending.

4. It is not desirable that your program is blocked when it is listening on the given port. Your program should always be listening on the given port while it can also allow user to send message. Modify the program in problem 3 to fulfill these purposes.

   Hint: You can modify `Messenger` class by extending the `Thread` class.

5. We recall the program in problem 4 of Exercise 2.3. Modify it by adding the `Messenger` class. Whenever a button a clicked, a message is sent to the target IP address and port number as follows (the target IP address and port number are entered by the user through the two text fields).

Clicked button	Message
Call	`INVITE` *`4000`*
Accept	`OK` *`5000`*
Disconnect	`BYE`
Remote call	`RINGING`
Remote accept	`ACK`
Remote disconnect	`OK`

   Test your program with the program that you wrote in problem 3.

6. Write a program named `PhoneSignaling` (i.e., the main class is `PhoneSignaling.java`) to implement the signaling protocol as described in Laboratory 1. Your program can accept two command-line arguments; one for the message port and other for the media port [see Fig. 2.3]. The default numbers of the message port and media port are 2000 and 4000, respectively. Hence, your program can be executed as follows.

	Message port	Media port
`java PhoneSignaling`	2000	4000
`java PhoneSignaling 3000`	3000	4000
`java PhoneSignaling 3000 5000`	3000	5000

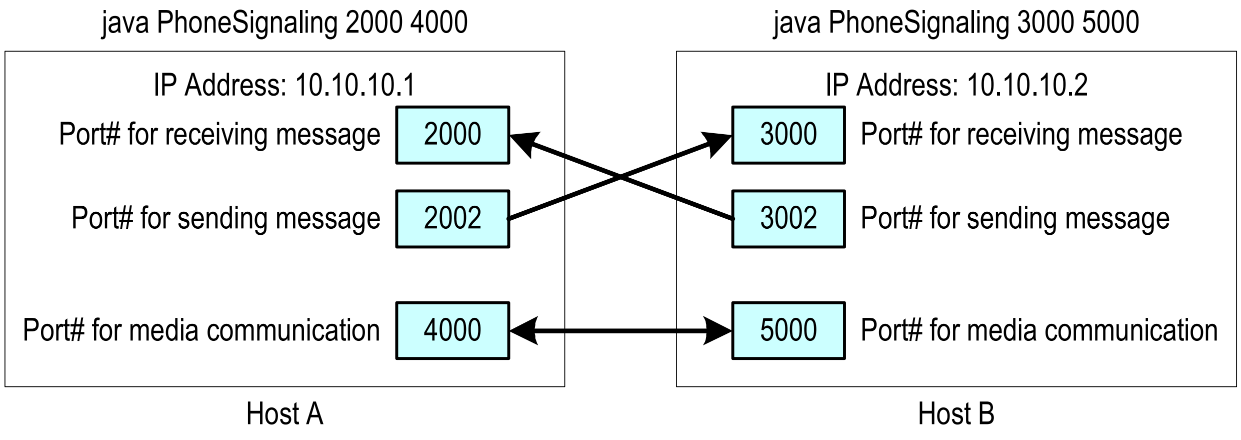

java PhoneSignaling 2000 4000
java PhoneSignaling 3000 5000
IP Address: 10.10.10.1
IP Address: 10.10.10.2
Port# for receiving message
2000
3000
Port# for receiving message
Port# for sending message
2002
3002
Port# for sending message
Port# for media communication
4000
5000
Port# for media communication
Host A
Host B

Fig. 2.3 Example of communication between two hosts.

Appendix

Some useful APIs are listed as follows.

Class: `InetAddress` `import java.net.*`

Description	It represents an Internet Protocol (IP) address.
Methods	**`static`** `InetAddress getByName(String host)` – Determines the IP address of a host, given the host's name. **Throws**: `UnknownHostException` – if no IP address for the host could be found.
	`static` `InetAddress getLocalHost()` – Returns the local host. **Throws**: `UnknownHostException` – if no IP address for the host could be found.
	`String getHostAddress()` – Returns the IP address string in textual presentation.

Class: `DatagramPacket` `import java.net.*`

Description	It represents a datagram packet. It is used to implement a connectionless packet delivery service. Each message is routed from one machine to another, based solely on information contained within the packet.
Constructors	`DatagramPacket(byte[] buf, int length, InetAddress host, int port)` – Creates a datagram packet in a `byte` array `buf` of the specified `length`. It also specifies the host and the port for which the packet is sent. This constructor is often used to create a packet for delivery.
	`DatagramPacket(byte[] buf, int length)` – Creates a datagram packet in a `byte` array `buf` of the specified length. This constructor is often used to create a packet for receiving
Methods	`byte[] getData()` – Returns the data buffer.
	`void setData(byte[] buf)` – Sets the data buffer for the packet.
	`int getLength()` – Returns the length of the data in the packet. This is not the length of the buffer.
	`InetAddress getAddress()` – Returns the IP address of the machine to which this datagram is being sent or from which the datagram was received.
	`void setAddress(InetAddress iaddr)` – Sets the IP address of the machine to which the datagram is being sent.
	`int getPort()` – Returns the port number on the remote host to which the datagram is being sent or from which the datagram was received.
	`void setPort(int port)` – Sets the port number on the remote host to which the datagram is being sent.

Class: `DatagramSocket` `import java.net.*`

Description	It represents a socket for sending and receiving datagram packets. It is the sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed.
Constructors	`DatagramSocket(int port)` – Creates a datagram socket which is bound with the specified port on the local host machine. **Throws**: `SocketException` – if the socket could not be opened, or the socket could not bind to the specified local port.

	`DatagramSocket()` – Creates a datagram socket which is bound with any available port on the local host machine. **Throws**: `SocketException` – if the socket could not be opened, or the socket could not bind to the specified local port.
Methods	`void send(DatagramPacket packet)` – Sends a datagram packet from this socket. **Throws**: `IOException` – if an I/O error occurs.
	`void receive(DatagramPacket packet)` – Receives a datagram packet from this socket. This method blocks until a datagram is received. **Throws**: `IOException` – if an I/O error occurs.
	`void close()` – Close this datagram socket. Any thread currently blocked by the corresponding `receive()` will throw a `SocketException`.

References

[1] J. F. Kurose and K. W. Ross, *Computer Networking A Top-Down Approach Featuring the Internet*, Addison Wesley, 2001.

[2] Y. D. Liang, *Introduction to Java Programming, Fourth Edition*, Prantice Hall, 2003.

[3] Java 2 Platform Standard Edition 5.0 API Specification.
[Online] Available: http://java.sun.com/j2se/1.5.0/docs/api/index.html.

[4] *Wikipedia The Free Encyclopedia*.
[Online] Available: http://www.wikipedia.org.