Redes de Computadores

Lab #1

Message-oriented Network Programming with UDP Sockets

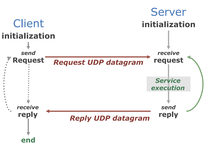
Summary ¶

Client/Server Model

Two autonomous components

- **Server** - first to run and usually always running
- **Client** - usually started by the user to request a service

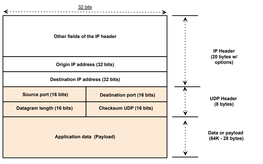
Client/Server Model with UDP Datagrams



What's a UDP Datagram?

- Raw byte sequence (at most 64K bytes long)
- **Addressed to** a host (**IP**) and a process (**port**)
- **From** a host (**IP**) and a process (**port**)
- Called an UDP datagram

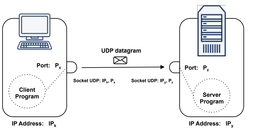
UDP Datagram Format



Communication using UDP Datagrams

- Based on **sockets**...
  - a *socket* provides an *end-point* abstraction with all the supported operations
- Addresses identify the sender and the receiver processes
  - *Host* (IP Address)
    - 10.1.233.67, 127.0.0.1, 192.168.1.1, etc.
  - *Port* (16 bits)
    - 8000

UDP Communication



Java Programming with UDP Datagrams

- Java package:
  - **java.net**
- Java classes
  - **DatagramSocket**
  - **DatagramPacket**
  - **InetAddress**

Example Application

ECHO

- Client sends a string message to the server (IP + port)
- Server echoes (returns) a copy of that string to the client

ECHO Server

```
final int PORT = 8000;
final int MAX_DATAGRAM_SIZE = 65536;

try(DatagramSocket socket = new DatagramSocket( PORT )) {
   for(;;) { // server endless loop
      // prepare an empty datagram
      byte[] buffer = new byte[MAX_DATAGRAM_SIZE];
      DatagramPacket echoRequest = new DatagramPacket( buffer, buffer.length );
      // wait for an incoming datagram
      socket.receive( echoRequest );

      //get UDP datagram payload
      byte[] echoRequestData = echoRequest.getData();
      int echoRequestLength = echoRequest.getLength();
      System.out.println( "GOT: "+ new String( echoRequestData, 0, echoRequestLength));

      // prepare an UDP datagram with the reply
      DatagramPacket echoReply = new DatagramPacket( echoRequestData, echoRequestLength ) ;
      // as well as destination IP address and port
      echoReply.setAddress( echoRequest.getAddress() );
      echoReply.setPort( echoRequest.getPort() );

      //send reply
      socket.send( echoReply );
   }
}
```

ECHO Client

```
static void main(String[] args) {
   if( args.length != 2 ) {
      System.out.println("usage: java EchoClient hostname port");
      System.exit(0);
   }

   // Prepare address and port of the server
   String server = args[0] ;
   int port = Integer.parseInt( args[1] );
   InetAddress srvAddress = InetAddress.getByName( server );
```

The **InetAddress** class is needed to convert a host name (string) into an IP address and store it...

```
String request;

// Create a helper scanner to read complete lines from standard input
try(Scanner in = new Scanner( System.in )) {
  // Read the "request" from the standard input
   System.out.print("Say what? " );
   request = in.nextLine();
}
```

```
// Prepare the socket to exchange datagrams
try( DatagramSocket socket = new DatagramSocket() ) {

  // Convert the request string into bytes
  byte[] reqData = request.getBytes();

  // Prepare the datagram filling in the contents and address in one go
  DatagramPacket echoReq = new DatagramPacket( reqData, reqData.length, srvAddress, port );

  socket.send( echoReq );

  // Prepare an empty datagram to receive the reply
  byte[] buffer = new byte[65536] ;
  DatagramPacket echoReply = new DatagramPacket( buffer, buffer.length );

  socket.receive( echoReply );

  // Create a string from the contents of the datagram
  String echoedMsg = new String( echoReply.getData(), 0, echoReply.getLength() ) ;
  System.out.printf("Got echo: \"%s\"\n", echoedMsg ) ;
}
```

ECHO Java Files

You can download the server and client as complete examples.

Recipes

Class DatagramSocket

```
DatagramSocket socket = new DatagramSocket()
socket.close()


DatagramSocket socket = new DatagramSocket( PORT )
socket.close()
```

Opened sockets consume resources... Must be closed after used.

```
try( DatagramSocket socket = new DatagramSocket() ) {
   ...
}

try( DatagramSocket socket = new DatagramSocket( PORT ) ) {
   ...
}
```

Since Java 7, **try with resources** blocks allow sockets to be closed automatically and release resources accordingly.

Class DatagramPacket

Receiving

```
byte[] buffer = new byte[MAX_DATAGRAM_SIZE];
DatagramPacket request = new DatagramPacket( buffer, buffer.length );

socket.receive( request );
```

Sending

```
DatagramPacket request = new DatagramPacket(msgData, msgData.length);
request.setAddress( srvAddress);
request.setPort( srvPort);
socker.send( request);


DatagramPacket request = new DatagramPacket(msgData, msgData.length, srvAddress, srvPort);
socker.send( request);
```

Class InetAddress

```
InetAddress myself = InetAddress.getByName("localhost");

InetAddress myself = InetAddress.getByName("127.0.0.1");

InetAddress myself = InetAddress.getLocalHost();

InetAddress server = InetAddress.getByName("www.wikipedia.org");

InetAddress server = InetAddress.getByName("200.10.78.9");

InetAddress[] servers = InetAddress.getAllByName("google.com");
```

Exercise

UDP File Transfer

Implement a client/server application capable of transferring files between two computers...

- A client program (**sender**) sends a file to a server (**receiver**) content to the server. Since UDP datagrams are limited in size, the contents of file will need to be split into multiple datagrams.
- Design the protocol so that:

  1. The transfer starts with a datagram containing the name of the file to be transferred;
  2. The contents of the file will be sent in blocks of at most 1024 bytes;
  3. The client and server can determine when the transfer has finished.

Java Tips

File I/O

- java.io.File - represents a file or folder in the filesystem. Can be used to query file attributes, including its length.
- java.io.FileOutputStream - for writing a file sequentially as a stream of bytes;
- java.io.FileInputStream - for reading a file sequentially as a stream of bytes. The stream read operation returns -1 when ***eof*** is reached.

Sleeping

- Thread.sleep(ms) - blocks the program the given time in milliseconds; Besides Facebook, it's a great time waster...

Timing Events/Operations

- The difference between two System.currentTimeInMillis() measurements provides the elapsed time in milliseconds;
- For more precision, System.nanoTime() measures time in nanoseconds (1 nanosecond = 1e-9 seconds)