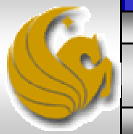
# COP 4610L: Applications in the Enterprise
# Spring 2005

## Java Networking and the Internet – Part 3

Instructor : Mark Llewellyn
markl@cs.ucf.edu
CSB 242, 823-2790
http://www.cs.ucf.edu/courses/cop4610L/spr2005

School of Electrical Engineering and Computer Science
University of Central Florida





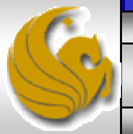
# More Details on Establishing a Server Using Stream Sockets

- Step 1 is to create a `ServerSocket` object.
- Invoking a `ServerSocket` constructor such as,

```
ServerSocket server =
    new ServerSocket (portNumber, queueLength);
```

registers an available TCP port number and specifies the number of clients that can wait to connect to the server (i.e., the queue length).

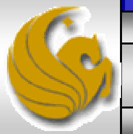




# More Details on Establishing a Server Using Stream Sockets (cont.)

- The port number is used by the clients to locate the server application on the server computer. This is often called the handshake point.
- If the queue is full, the server refuses client connections.
- The constructor establishes the port where the server waits for connections from clients – a process known as binding the server to the port.
- Each client will ask to connect to the server on this port. Only one application at a time can be bound to a specific port on the server.

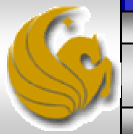




# More Details on Establishing a Server Using Stream Sockets (cont.)

- Port numbers can be between 0 and 65,535. Most OS reserve port numbers below 1024 for system services such as email, and Internet servers. Generally, these ports should not be specified as connection ports in user programs. In fact, some OS require special access privileges to bind to port numbers below 1024.
- Programs manage each client connection with a `Socket` object.





# More Details on Establishing a Server Using Stream Sockets (cont.)

- In Step 2, the server listens indefinitely (is said to block) for an attempt by a client to connect. To listen for a client connection, the program calls `ServerSocket` method `accept`, as in,

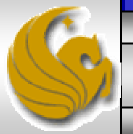
  ```
  Socket connection = server.accept();
  ```

  which returns a Socket when a connection with a client is established.

- The Socket allows the server to interact with the client.

- The interactions with the client actually occur at a different server port from the handshake port. This allows the port specified in Step 1 to be used again in a multi-threaded server to accept another client connection. We'll see an example of this later in this set of notes.

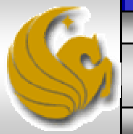




# More Details on Establishing a Server Using Stream Sockets (cont.)

- In Step 3, the `OutputStream` and `InputStream` objects that enable the server to communicate with the client by sending and receiving bytes are established.
- The server sends information to the client via an `OutputStream` and received information from the client via an `InputStream`.
- The server invokes the method `getOutputStream` on the `Socket` to get a reference to the Socket's `OutputStream` and invokes method `getInputStream` on the `Socket` to get a reference to the Socket's `InputStream`.





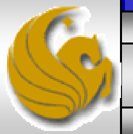
# More Details on Establishing a Server Using Stream Sockets (cont.)

- If primitive types or serializable types (like String) need to be sent rather than bytes, wrapper classes are used to wrap other stream types around the `OutputStream` and `InputStream` objects associated with the `Socket`.

```
ObjectInputStream input =
    new(ObjectInputStream(connection.getInputStream());
```

```
ObjectOutputStream output =
    new(ObjectOutputStream(connection.getOutputStream());
```

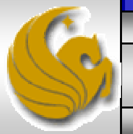




# More Details on Establishing a Server Using Stream Sockets (cont.)

- The beauty of establishing these relationships is that whatever the server writes to the `ObjectOutputStream` is set via the `OutputStream` and is available at the client's `InputStream`, and whatever the client writes to its `OutputStream` (with a corresponding `ObjectOutputStream`) is available via the server's `InputStream`.
- The transmission of the data over the network is seamless and is handled completely by Java.





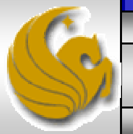
# More Details on Establishing a Server Using Stream Sockets (cont.)

- With Java's multithreading, you can create multithreaded servers that can manage many simultaneous connections with many clients.
- A multithreaded server can take the `Socket` returned by each call to accept and create a new thread that manages network I/O across that `Socket`.
  - Alternatively, a multithreaded sever can maintain a pool of threads (a set of already existing threads) ready to manage network I/O across the new `Sockets` as they are created. In this fashion, when the server receives a connection, it need not incur the overhead of thread creation. When the connection is closed, the thread is returned to the pool for reuse.

# More Details on Establishing a Server Using Stream Sockets (cont.)

- Step 4 is the processing phase, in which the server and client communicate via the `OutputStream` and `InputStream` objects.
- In Step 5, when the transmission is complete, the server closes the connection by invoking the `close` method on the streams and on the `Socket`.





# More Details on Establishing a Client Using Stream Sockets

- Step 1 is to create a `Socket` object to connect to the server. The Socket constructor established the connection with the server.

- For example, the statement

```
Socket connection = new Socket(serverAddress, port);
```

uses the `Socket` constructor with two arguments – the server's address and the port number.

- If the connection attempt is successful, this statement returns a `Socket`.

# More Details on Establishing a Client Using Stream Sockets (cont.)

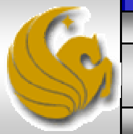
- If the connection attempt fails, an instance of a subclass of IOException, since so many program simply catch IOException.
- An UnknownHostException occurs specifically when the system is unable to resolve the server address specified in the call to the Socket constructor to a corresponding IP address.

# More Details on Establishing a Client Using Stream Sockets (cont.)

- In Step 2, the client uses Socket methods `getInputStream` and `getOutputStream` to obtain references to the `Socket`'s `InputStream` and `OutputStream`.

- If the server is sending information in the form of actual types (not byte streams) the client should receive the information in the same format. Thus, if the server sends values with an `ObjectOutputStream`, the client should read those values with an `ObjectInputStream`.

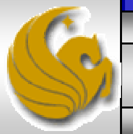




# More Details on Establishing a Client Using Stream Sockets (cont.)

- Step 3 is the same as in the server, where the client and the server communicate via `InputStream` and `OutputStream` objects.
- In Step 4, the client closes the connection when the transmission is complete by invoking the close method on the streams and on the Socket.
- The client must determine when the server is finished sending information so that it can call `close` to close the Socket connection.
- For example, the `InputStream` method read returns the value -1 when it detects end-of-stream (also called EOF). If an `ObjectInputStream` is used to read information from the server, an `EOFException` occurs when the client attempts to read a value from a stream on which end-of-stream is detected.

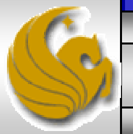




# Secure Sockets Layer (SSL)

- Most e-business uses SSL for secure on-line transactions.
- SSL does not explicitly secure transactions, but rather secures connections.
- SSL implements public-key technology using the RSA algorithm (developed in 1977 at MIT by Ron Rivest, Adi Shamir, and Leonard Adleman) and digital certificates to authenticate the server in a transaction and to protect private information as it passes from one part to another over the Internet.
- SSL transaction do not require client authentication as most servers consider a valid credit-card number to be sufficient for authenticating a secure purchase.

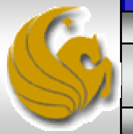




# How SSL Works

- Initially, a client sends a message to a server.
- The server responds and sends its digital certificate to the client for authentication.
- Using public-key cryptography to communicate securely, the client and server negotiate session keys to continue the transaction.
- Once the session keys are established, the communication proceeds between the client and server using the session keys and digital certificates.
- Encrypted data are passed through TCP/IP (just as regular packets over the Internet). However, before sending a message with TCP/IP, the SSL protocol breaks the information into blocks and compresses and encrypts those blocks.

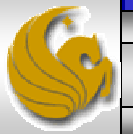




# How SSL Works (cont.)

- Once the data reach the receiver through TCP/IP, the SSL protocol decrypts the packets, then decompresses and assembles the data. It is these extra processes that provide an extra layer of security between TCP/IP and applications.
- SSL is used primarily to secure point-to-point connections using TCP/IP rather than UDP/IP.
- The SSL protocol allows for authentication of the server, the client, both, or neither. Although typically in Internet SSL sessions only the server is authenticated.

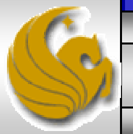




# Java Secure Socket Extension (JSSE)

- SSL encryption has been integrated into Java technology through the Java Secure Socket Extension (JSSE). JSSE has been an integral part of Java (not a separately loaded extension) since version 1.4.
- JSSE provides encryption, message integrity checks, and authentication of the server and client.
- JSSE uses keystores to secure storage of key pairs and certificates used in PKI (Public Key Infrastructure which integrates public-key cryptography with digital certificates and certificate authorities to authenticate parties in a transaction.)
- A truststore is a keystore that contains keys and certificates used to validate the identities of servers and clients.

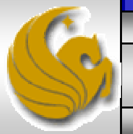




# Java Secure Socket Extension (JSSE) (cont.)

- Using secure sockets in Java is very similar to using the non-secure sockets that we have already seen.
- JSSE hides the details of the SSL protocol and encryption from the programmer entirely.
- The final example in this set of notes involves a client application that attempts to logon to a server using SSL.
- **NOTE:** Before attempting to execute this application, look at the code first and then go to page 25 for execution details. This application will not execute correctly unless you follow the steps beginning on page 25.





```java
// LoginServer.java
// LoginServer uses an SSLServerSocket to demonstrate JSSE's SSL implementation.
package securitystuff.jsse;

// Java core packages
import java.io.*;

// Java extension packages
import javax.net.ssl.*;

public class LoginServer {
   private static final String CORRECT_USER_NAME = "Mark";
   private static final String CORRECT_PASSWORD = "COP 4610L";
   private SSLServerSocket serverSocket;

   // LoginServer constructor
   public LoginServer() throws Exception
   {
      // SSLServerSocketFactory for building SSLServerSockets
      SSLServerSocketFactory socketFactory =
         ( SSLServerSocketFactory )
            SSLServerSocketFactory.getDefault();
      // create SSLServerSocket on specified port
      serverSocket = ( SSLServerSocket )
         socketFactory.createServerSocket( 7070 );

   } // end LoginServer constructor
```

LoginServer.java

SSL Server Implementation

Use default SSLServerSocketFactory to create SSL sockets

SSL socket will listen on port 7070

```java
// start server and listen for clients
private void runServer()
{
   // perpetually listen for clients
   while ( true ) {
     // wait for client connection and check login information
     try {
          System.err.println( "Waiting for connection..." );
           // create new SSLSocket for client
          SSLSocket socket =  ( SSLSocket ) serverSocket.accept();
         // open BufferedReader for reading data from client
         BufferedReader input = new BufferedReader(
            new InputStreamReader( socket.getInputStream() ) );
         // open PrintWriter for writing data to client
        PrintWriter output = new PrintWriter(
          new OutputStreamWriter(socket.getOutputStream() ) );
        String userName = input.readLine();
        String password = input.readLine();
         if ( userName.equals( CORRECT_USER_NAME ) &&
            password.equals( CORRECT_PASSWORD ) ) {
          output.println( "Welcome, " + userName );
       }
       else {
          output.println( "Login Failed." );
       }
```

Accept new client connection. This is a blocking call that returns an SSLSocket when a client connects.

Get input and output streams just as with normal sockets.

Validate user name and password against constants on the server.

```
        // clean up streams and SSLSocket
        output.close();
        input.close();
        socket.close();

      } // end try

      // handle exception communicating with client
      catch ( IOException ioException ) {
        ioException.printStackTrace();
      }

    } // end while

  } // end method runServer

  // execute application
  public static void main( String args[] ) throws Exception
  {
    LoginServer server = new LoginServer();
    server.runServer();
  }
} //end LoginServer class
```

Close down I/O streams and the socket

```java
// LoginClient.java
// LoginClient uses an SSLSocket to transmit fake login information to LoginServer.
package securitystuff.jsse;
// Java core packages
import java.io.*;
// Java extension packages
import javax.swing.*;
import javax.net.ssl.*;

public class LoginClient {
   // LoginClient constructor
  public LoginClient()
  {
     // open SSLSocket connection to server and send login
     try {
        // obtain SSLSocketFactory for creating SSLSockets
        SSLSocketFactory socketFactory =  ( SSLSocketFactory ) SSLSocketFactory.getDefault();
        // create SSLSocket from factory
        SSLSocket socket =   ( SSLSocket ) socketFactory.createSocket(  "localhost", 7070 );
        // create PrintWriter for sending login to server
        PrintWriter output = new PrintWriter(
           new OutputStreamWriter( socket.getOutputStream() ) );
         // prompt user for user name
        String userName = JOptionPane.showInputDialog( null, "Enter User Name:" );
         // send user name to server
        output.println( userName );
```

LoginClient.java

Client Class for SSL Implementation

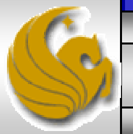
Use default SSLServerSocketFactory to create SSL sockets

SSL socket will listen on port 7070

```
      // prompt user for password
      String password = JOptionPane.showInputDialog( null, "Enter Password:" );
      // send password to server
      output.println( password );
      output.flush();
       // create BufferedReader for reading server response
      BufferedReader input = new BufferedReader(
         new InputStreamReader( socket.getInputStream () ) );
       // read response from server
      String response = input.readLine();
      // display response to user
      JOptionPane.showMessageDialog( null, response );
       // clean up streams and SSLSocket
      output.close();
      input.close();
      socket.close();
    } // end try
   // handle exception communicating with server
   catch ( IOException ioException ) {
      ioException.printStackTrace();
   }
   // exit application
   finally {
      System.exit( 0 );
   }
} // end LoginClient constructor
```

```
// execute application
   public static void main( String
args[] )
   {
      new LoginClient();
   }
}
```

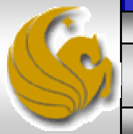




# Creating Keystore and Certificate

- Before you can execute the LoginServer and LoginClient application using SSL you will need to create a keystore and certificate for the SSL to operate correctly.
- Utilizing the keytool (a key and certificate management tool) in Java generate a keystore and a certificate for this server application. See the next slide for an example.
- We'll use the same keystore for both the server and the client although in reality these are often different. The client's truststore, in real-world applications, would contain trusted certificates, such as those from certificate authorities (e.g. VeriSign (www.verisign.com), etc.).





# Creating Keystore and Certificate

Command Prompt (2)

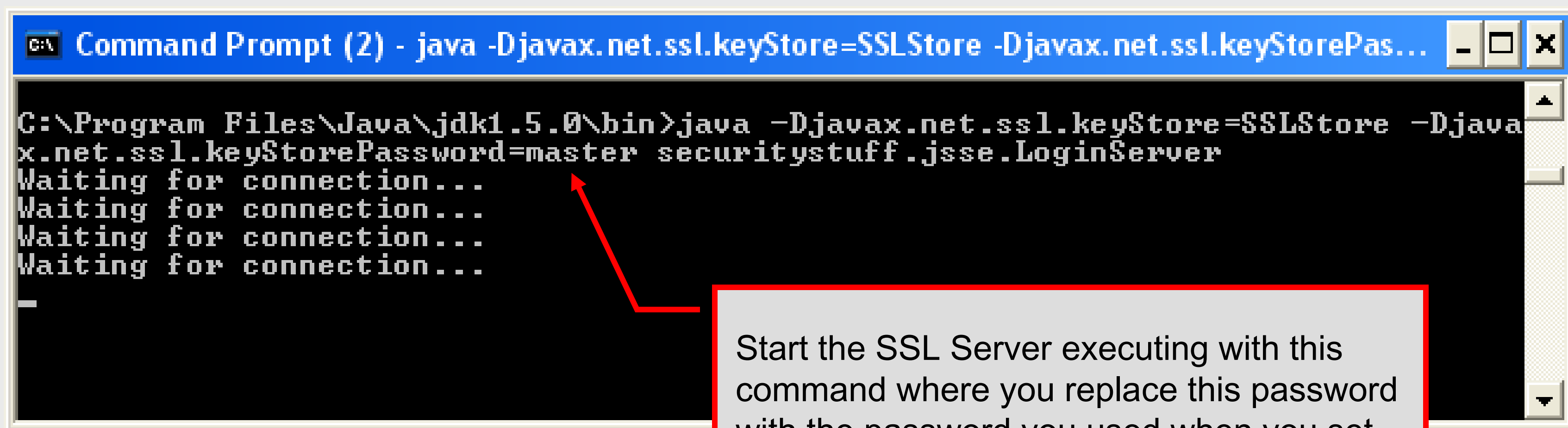
```
C:\Program Files\Java\jdk1.5.0\bin>keytool -genkey -keystore SSLStore -alias SSL
Certificate
Enter keystore password:  root
Keystore password is too short - must be at least 6 characters
Enter keystore password:  master
What is your first and last name?
  [Unknown]:  Mark Llewellyn
What is the name of your organizational unit?
  [Unknown]:  UCF CS
What is the name of your organization?
  [Unknown]:  UCF
What is the name of your City or Locality?
  [Unknown]:  Orlando
What is the name of your State or Province?
  [Unknown]:  Florida
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=Mark Llewellyn, OU=UCF CS, O=UCF, L=Orlando, ST=Florida, C=US correct?
  [no]:  yes

Enter key password for <SSLCertificate>
        (RETURN if same as keystore password):  master

C:\Program Files\Java\jdk1.5.0\bin>
```

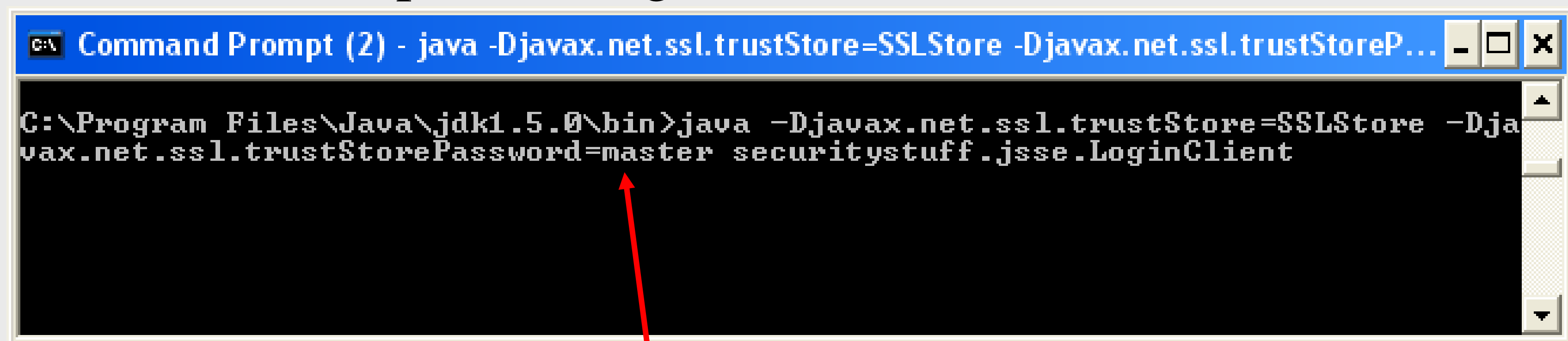
# Launching the Secure Server

- Start the server executing from a command prompt…
- Once started, the server simply waits for a connection from a client. The example below illustrates the server after waiting for several minutes.

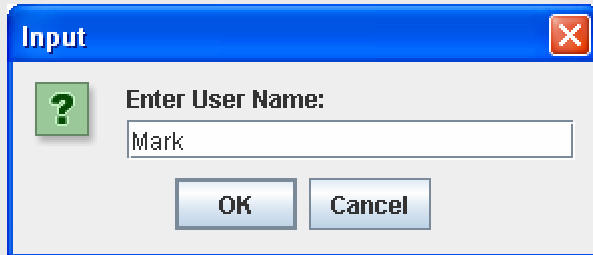


# Launching the SSL Client

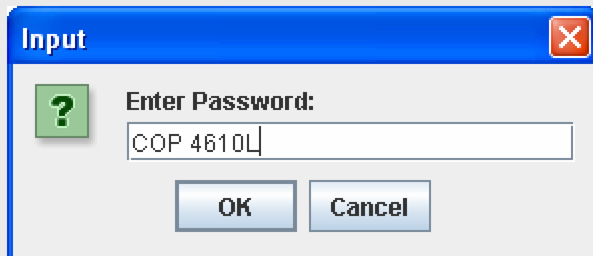
- Start a client application executing from a new command window…
- Once the client establishes communication with the server, the authentication process begins.



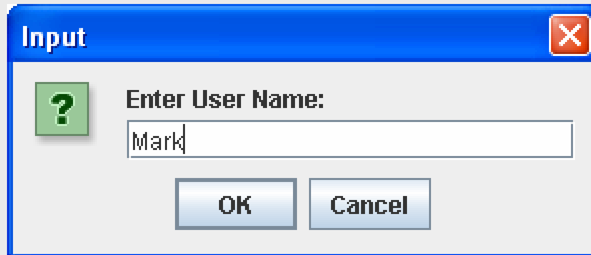

Start the SSL Client application executing with this command where you replace this password with the password you used when you set-up the keystore. Since we are using the same keystore for the server and the client…these will be the same.

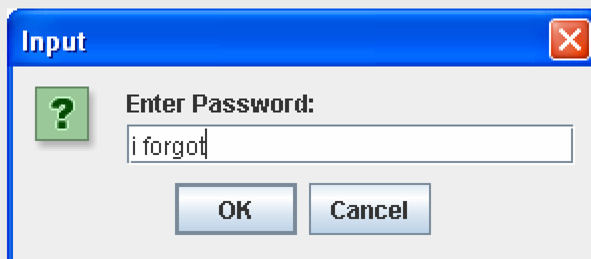



User enters username and password which are sent to the server.

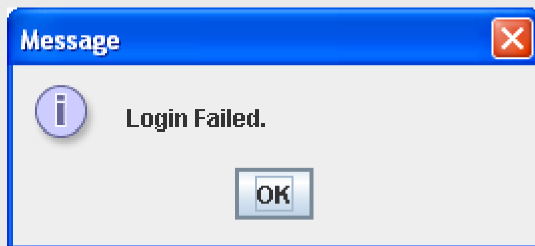



Authentication successful – user is logged on.




User enters username and password which are sent to the server. In this case the user enters an incorrect password.

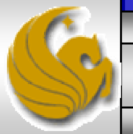


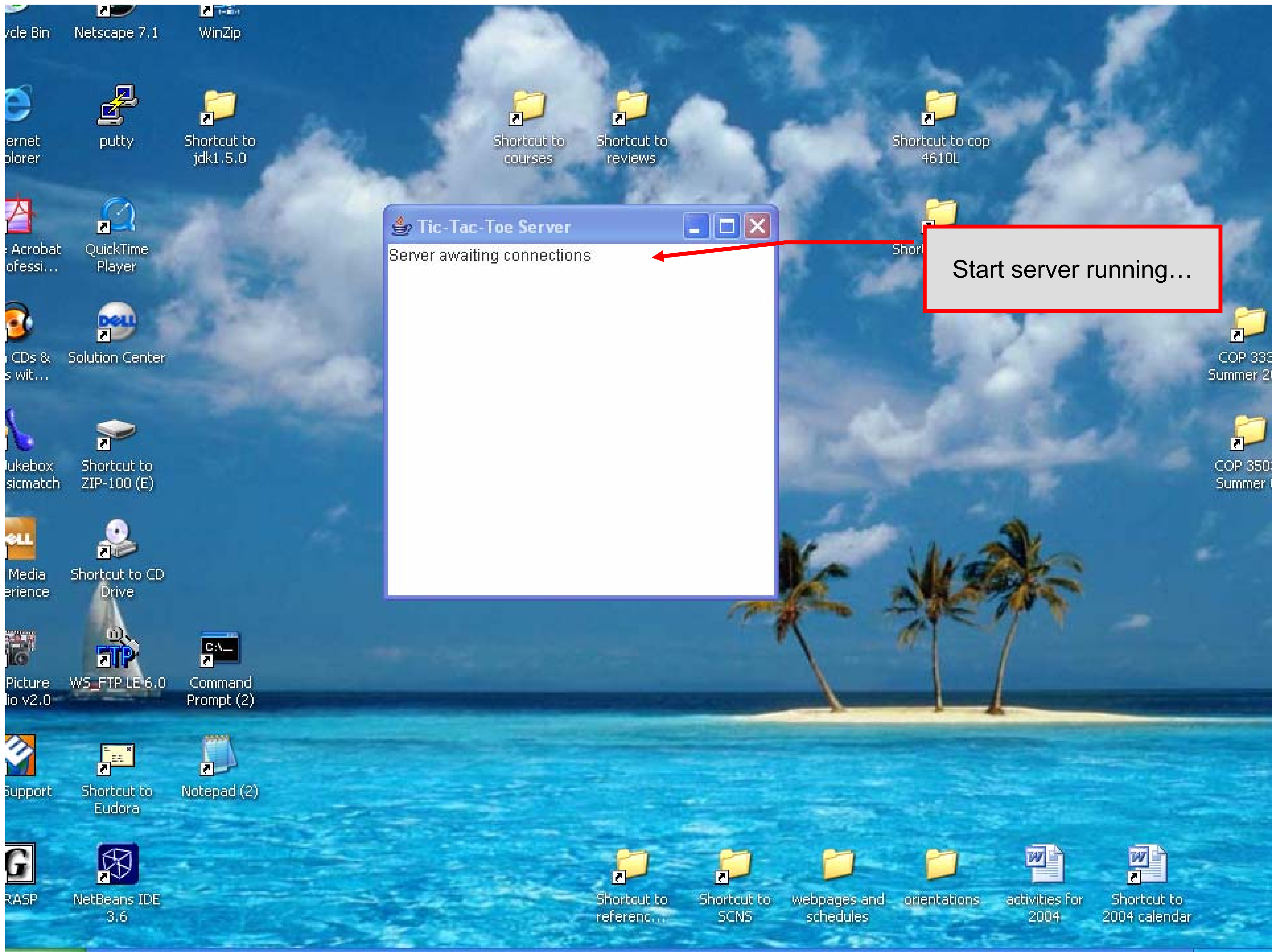

Authentication not successful – user is not logged on.

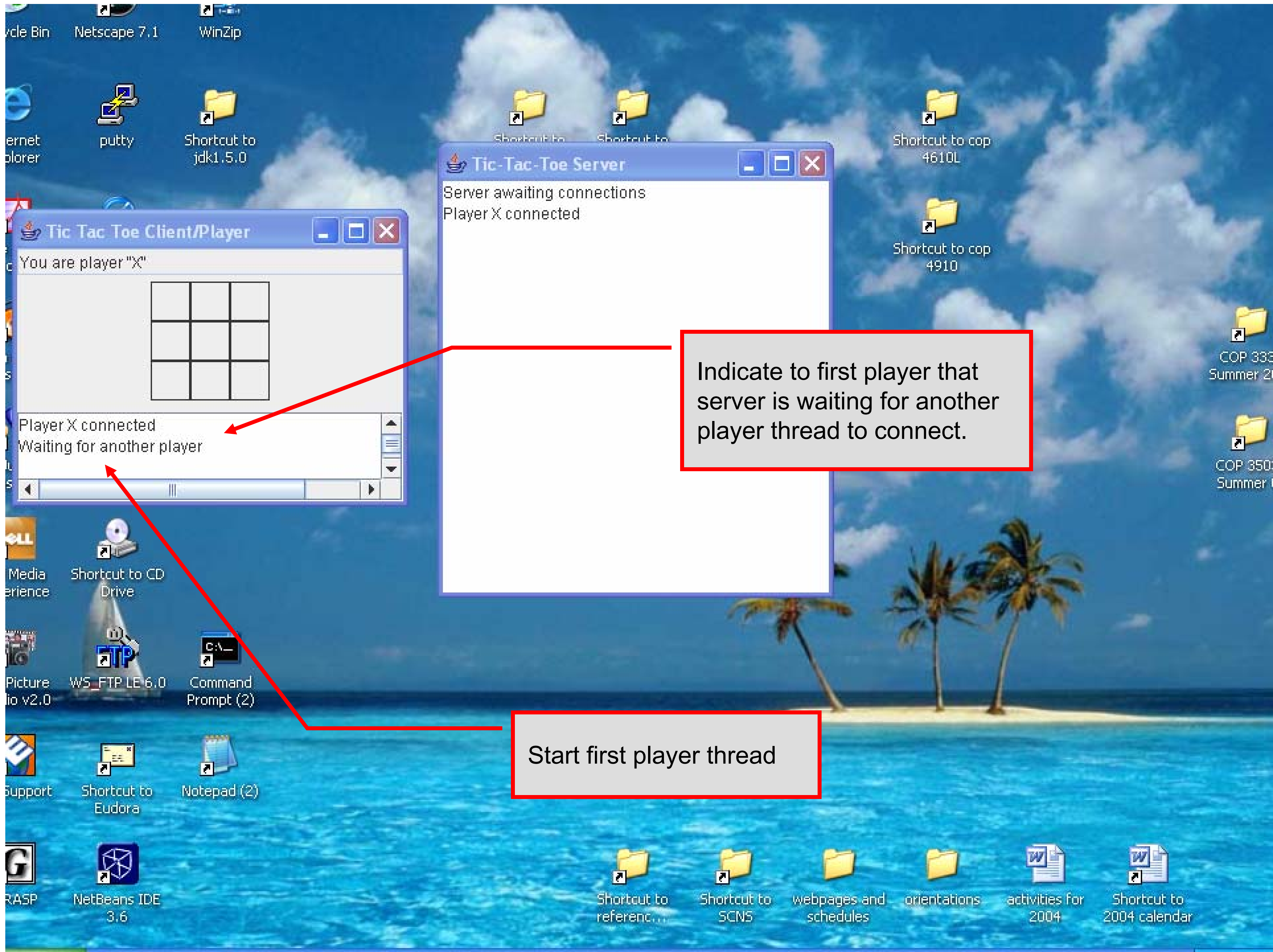
# Multithreaded Socket Client/Server Example

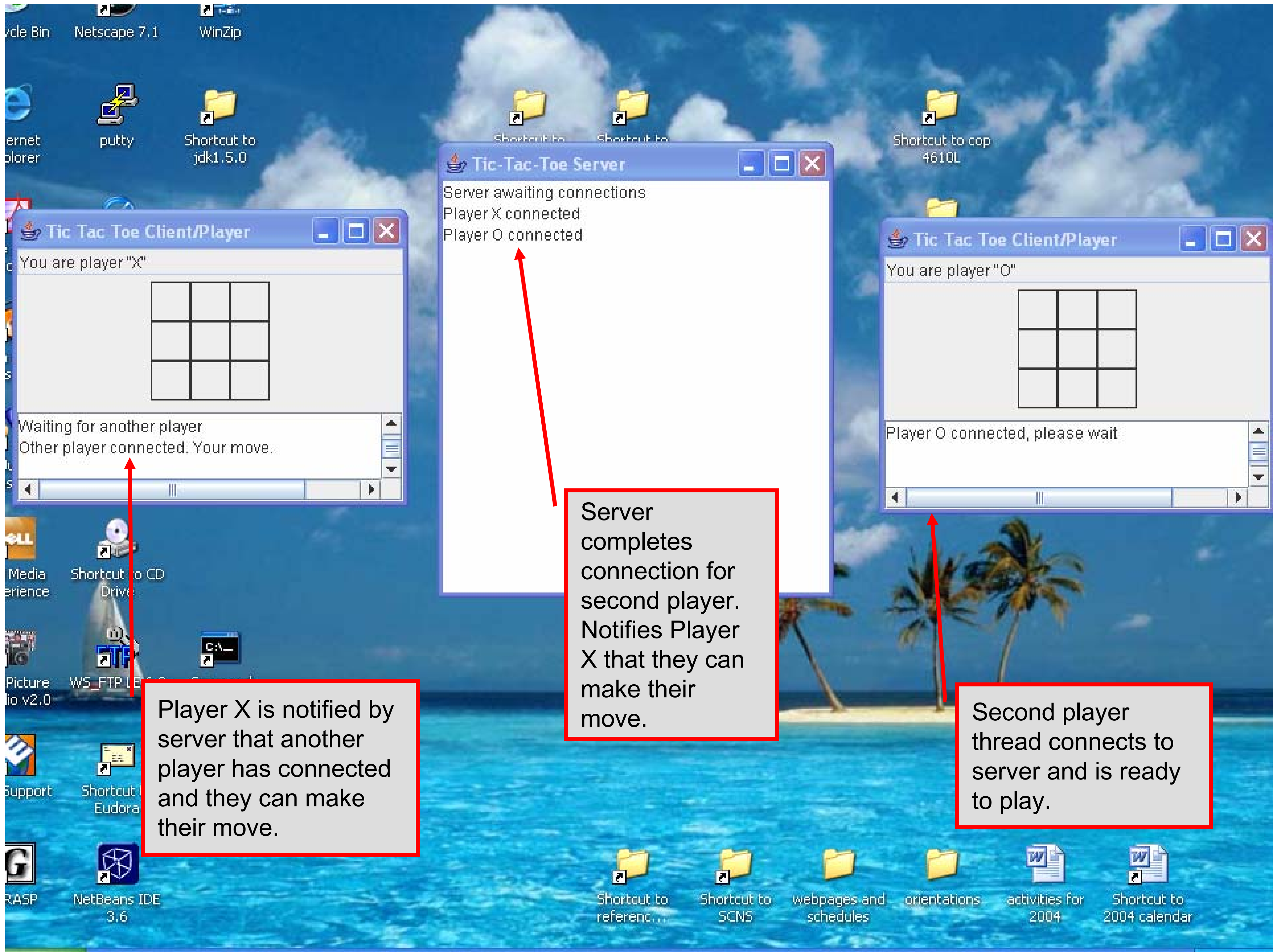
- As a culminating example of networking and multi-threading, I've put together a rudimentary multi-threaded socket-based TicTacToe client/server application. The code is rather lengthy and there isn't really anything in it that we haven't already seen in the earlier sections of the notes. However, I did want you to see a somewhat larger example that utilizes both sockets and threading in Java. The code is on the course web page so try it out.
- This application is a multithreaded server that will allow two client's to play a game of TicTacToe run on the server.
- To execute, open three command windows, start one server and two clients (in separate windows).
- The following few pages contain screen shots of what you should see when executing this code.

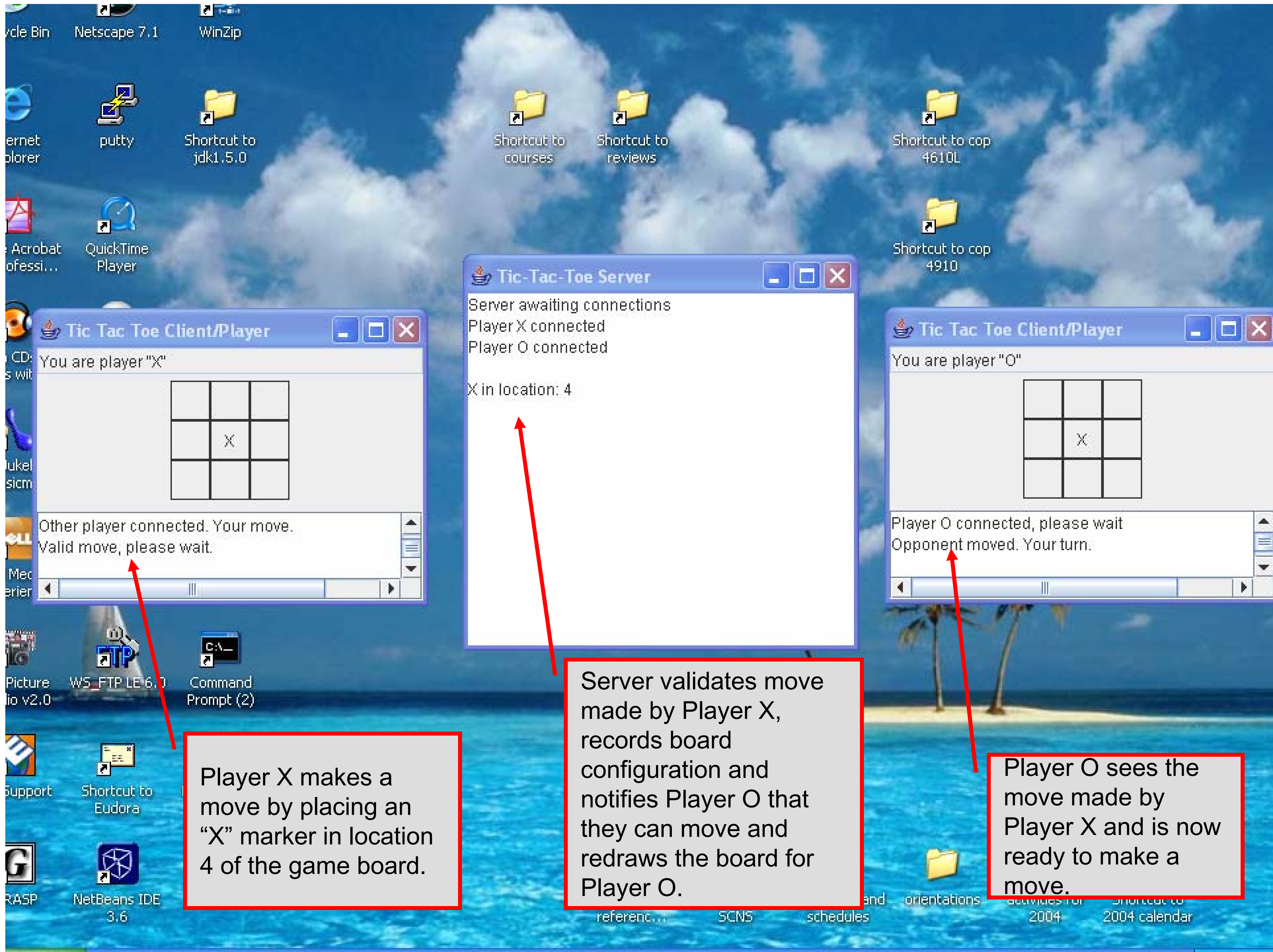




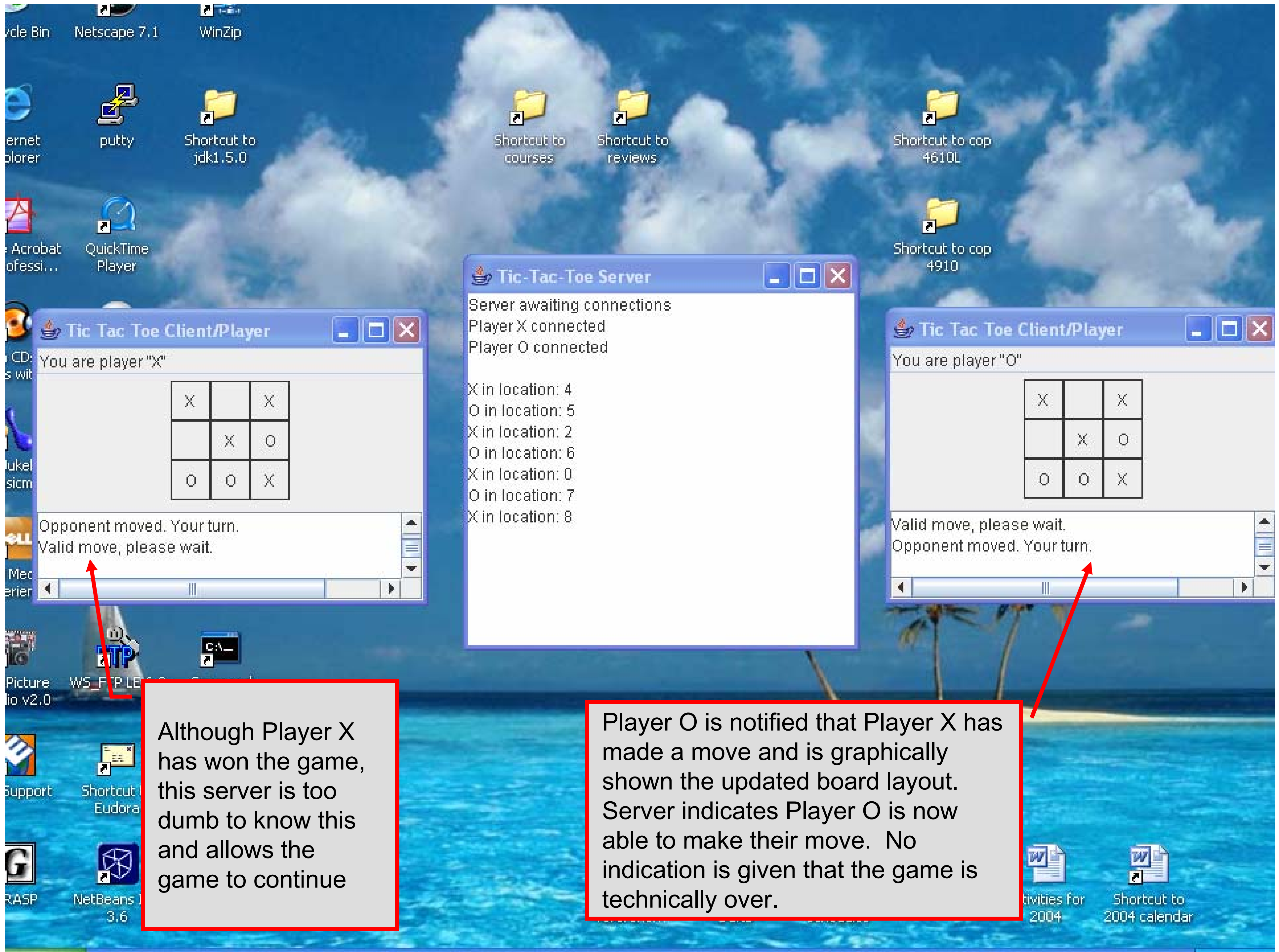

Tic-Tac-Toe Server
Server awaiting connections
Start server running…


Tic-Tac-Toe Server
Server awaiting connections
Player X connected
Tic Tac Toe Client/Player
You are player "X"
Player X connected
Waiting for another player
Indicate to first player that server is waiting for another player thread to connect.
Start first player thread


Tic-Tac-Toe Server
Server awaiting connections
Player X connected
Player O connected
Tic Tac Toe Client/Player
You are player "X"
Waiting for another player
Other player connected. Your move.
Tic Tac Toe Client/Player
You are player "O"
Player O connected, please wait
Server completes connection for second player. Notifies Player X that they can make their move.
Player X is notified by server that another player has connected and they can make their move.
Second player thread connects to server and is ready to play.



Netscape 7.1
WinZip
putty
Shortcut to jdk1.5.0
Shortcut to courses
Shortcut to reviews
Shortcut to cop 4610L
QuickTime Player
Shortcut to cop 4910
Tic-Tac-Toe Server
Server awaiting connections
Player X connected
Player O connected
X in location: 4
Tic Tac Toe Client/Player
You are player "X"
X
Other player connected. Your move.
Valid move, please wait.
Tic Tac Toe Client/Player
You are player "O"
X
Player O connected, please wait
Opponent moved. Your turn.
WS_FTP LE 6.0
Command Prompt (2)
Shortcut to Eudora
NetBeans IDE 3.6
orientations
schedules
2004 calendar
Player X makes a move by placing an “X” marker in location 4 of the game board.
Server validates move made by Player X, records board configuration and notifies Player O that they can move and redraws the board for Player O.
Player O sees the move made by Player X and is now ready to make a move.



Tic Tac Toe Client/Player
You are player "X"
Opponent moved. Your turn.
Valid move, please wait.
Tic-Tac-Toe Server
Server awaiting connections
Player X connected
Player O connected
X in location: 4
O in location: 5
X in location: 2
O in location: 6
X in location: 0
O in location: 7
X in location: 8
Tic Tac Toe Client/Player
You are player "O"
Valid move, please wait.
Opponent moved. Your turn.
Although Player X has won the game, this server is too dumb to know this and allows the game to continue
Player O is notified that Player X has made a move and is graphically shown the updated board layout. Server indicates Player O is now able to make their move. No indication is given that the game is technically over.