Milwaukee School of Engineering
CS496 – Networking Protocols
Lab2 – Sockets

Student: Marek Handl
Date: January 11th 2006

➢ IMPLEMENTATION

Java Threads were used to achieve desired functionality.
Main server thread (class Server) listens on given port. Once a client connects a new thread (class ServerThread) is created that will handle all the communication. The server has no possibility of user's input, it just displays incoming messages and connection status.
Client composes of two threads. The main one (class Client) establishes connection, reads user's input and send it to server. The other thread (class ClientReadThread) reads from socket, i.e. it reads possible echo from server.
Using commands on the client's side, it is possible to close server's output, client's input, the whole connection and to switch on/off the echo from the server.

➢ SOURCE CODE

```
/*
 * Server
 * listens on selected port and creates new threads for each client that connects
 */
public class Server {

        public static void main(String[] args) {

                // implicit parameters
                int port = 1234;
                boolean echo = false;

                // parsing of command line parameters
                if ( args.length > 0 ) {
                        if ( args[0].equals("-h") ) {
                                System.out.println("Usage: ServerMulti port_number echo      | echo=on ... echo is on, otherwise is off");
                                System.exit(0);
                        }
                        else {
                                try {
                                        port = Integer.parseInt( args[0] );
                                        if ( args.length > 1 && args[1].equals("on") )
                                                echo = true;
                                }
                                catch (Exception e) {
                                        System.out.println( "Wrong parameters!" );
                                        System.out.println( "Usage: ServerMulti port_number echo      .... 1 is echo_on");
                                        System.exit(0);
                                }
                        }
                }
                boolean alwaysTrue = true;
```

```java
                try {

                        // start listening
                        ServerSocket srvsoc = new ServerSocket( port );
                        System.out.println( "Server running." );
                        System.out.println( "Server IP address: " + InetAddress.getLocalHost() );
                        System.out.println( "Listening on port: " + port );
                        System.out.print( "Echo is " );
                        if ( echo )
                                System.out.println( "ON.\n" );
                        else
                                System.out.println( "OFF.\n" );

                        // wait for clients to connect and create a new thread for each one
                        while ( alwaysTrue ) {

                                new ServerThread( srvsoc.accept(), echo ).start();

                        }

                        // stop listening and exit
                        srvsoc.close();
                        System.out.println( "Server shut down" );

                }
                catch (Exception e ) {
                        System.out.println( "Something went wrong - server!!! " + e.getMessage() );
                }

        }
}


/* ************************************************************************************************** */
/*
 * part of Server program
 * each thread handles communication with one client
 * it can echo back the incoming messages
 */
public class ServerThread extends Thread {

        private Socket socket;
        private boolean echo = true;        // echo incoming messages back to client?
        private boolean clientClosedHisInput = false;

        // constructor
        public ServerThread( Socket socket, boolean echo ) {
                super("ServerThread");
                this.socket = socket;
                this.echo = echo;
        }

        // main thread function
        public void run() {

                try {

                        System.out.println( "Client connected: " + socket.getInetAddress() + ":" + socket.getPort() );
                        PrintWriter out = new PrintWriter( socket.getOutputStream(), true );              // to be able to write to a socket
                                                                                                          // (necessary for echoing back)
                        BufferedReader in = new BufferedReader( new InputStreamReader( socket.getInputStream() ));      // to be
                                                                                                          // able to read from a socket

                        String inputLine;

                        // read incoming messages
                        while( (inputLine = in.readLine()) != null ) {

                                // user commands
                                if ( inputLine.equals("END_CONNECTION") )           // end this thread
                                        break;
```

```java
                                            if ( inputLine.equals( "ECHO_OFF" ) )
                                                        echo = false;
                                            if ( inputLine.equals( "ECHO_ON" ) )
                                                        echo = true;
                                            if ( inputLine.equals( "SHUT_INPUT" ) ) {
                                                        // half-closed state, client closed his input
                                                        // no echo anymore - it would cause an exception
                                                        clientClosedHisInput = true;
                                            }
                                            if ( inputLine.equals( "SHUT_SERV_OUTPUT" ) ) {
                                                        // half-closed state, server shuts his output
                                                        // even with echo enabled, nothing will be echoed back to the client
                                                        socket.shutdownOutput();
                                                        System.out.println( "No more output for " + socket.getInetAddress() + ":" +
                                                                                                socket.getPort() );
                                            }

                                            // display incoming message
                                            System.out.println( socket.getInetAddress() + ":" + socket.getPort() + "  :  " + inputLine );

                                            // echo the message back to the client
                                            if ( echo && !clientClosedHisInput ) {
                                                        out.println( inputLine );
                                                        out.flush();
                                            }
                                }

                                // end connection and exit
                                System.out.println( "Client disconnected " + socket.getInetAddress() + ":" + socket.getPort());
                                out.close();
                                in.close();
                                socket.close();                 // this ends the connection

                    }
                    catch (Exception e ) {
                                System.out.println( "Something went wrong - server thread!!! " + e.getMessage() );
                    }

          }

}

/* ********************************************************************************************* */

/*
 * Client
 * connects to given server
 * reads users input and sends it to server - this thread
 * creates one thread for reading messages sent by server
 */
public class Client {

          public static void main(String[] args) {

                    // implicit parameters
                    int port = 1234;
                    String ip = "localhost";

                    // parsing of command line parameters
                    if ( args.length > 0 ) {
                                if ( args[0].equals("-h") ) {
                                            System.out.println( "Usage: Client server_ip server_port");
                                            System.out.println( "User Commands:\n" +
                                                        " END_CONNECTION ... close the socket and exit\n" +
                                                        " ECHO_ON ... let the server echo back everything I send\n" +
                                                        " ECHO_OFF ... the server will not echo back messages\n" +
                                                        " SHUT_INPUT ... close my socket input - no echo will be received\n" +
                                                    " SHUT_SERV_OUTPUT ... close server's output - no echo will be received" );
                                            System.exit(0);
                                }
```

```java
                                else {
                                        try {
                                                ip = args[0];
                                                port = Integer.parseInt( args[1] );
                                        }
                                        catch (Exception e) {
                                                System.out.println( "Wrong parameters!" );
                                                System.out.println( "Usage: Client server_ip server_port");
                                                System.exit(0);
                                        }
                                }
                        }
                        else {
                                System.out.println( "Using implicit parameters: ip=localhost, port=1234.");
                        }

                        System.out.println( "Trying to connect..." );


                        try {

                                // socket set-up
                                Socket socket = new Socket( ip, port );         // connect to server
                                System.out.println( "Connected to " + ip + ":" + socket.getPort() + "    local port: " + socket.getLocalPort() );
                                Thread readingThread = new ClientReadThread( socket );
                                readingThread.start(); // new thread will read and display messages from server
                                BufferedReader userIn = new BufferedReader( new InputStreamReader( System.in ) );    // user input
                                PrintWriter out = new PrintWriter( socket.getOutputStream(), true );            // socket output
                                String userLine;

                                // read user's input and send it to server
                                while ( (userLine = userIn.readLine()) != null ) {

                                        out.println( userLine );                // send message
                                        out.flush();

                                        // user commands
                                        if ( userLine.equals( "END_CONNECTION" ) )    // users command to close the connection and exit
                                                break;
                                        if ( userLine.equals( "SHUT_INPUT" ) ) { // user command to stop reading from socket
                                                readingThread.interrupt(); // finish own reading thread
                                                Thread.sleep(100);    // give the reading thread chance to finish
                                                socket.shutdownInput();
                                                System.out.println( "Socket input is shut down.");
                                        }

                                }

                                // close connection and exit
                                System.out.println( "Closing..." );
                                readingThread.interrupt();             // finish own reading thread
                                Thread.sleep(100);    // give the reading thread chance to finish
                                userIn.close();
                                out.close();
                                socket.close();
                                System.out.println( "Connection closed" );

                        }
                        catch (Exception e ) {
                                System.out.println( "Something went wrong!!! " + e.getMessage() );
                        }

                }

}

/* ********************************************************************************************** */
```

```
/*
 * part of the Client program
 * reads data from socket and displays them on screen
 */
public class ClientReadThread extends Thread {

        private Socket socket;

        // constructor
        public ClientReadThread( Socket socket ) {
                    super("ClientReadThread");
                    this.socket = socket;
        }

        // main function
        public void run() {

                try {

                            // this is how to read from a socket
                            BufferedReader in = new BufferedReader( new InputStreamReader( socket.getInputStream() ));

                            while ( !in.ready() ) {

                                        if ( Thread.interrupted() ) {       // main thred wants to close this thread, because of half-closed state
                                                                            // or the connection is being closed
                                                    break;
                                        }
                                        String line = in.readLine();
                                        if ( line != null ) // necessary at the moment of socket closing
                                                    System.out.println( line );

                            }

                            // if the socket is still open, close reading stream
                            try{
                                        in.close();
                            }catch (Exception e){}

                }
                catch (Exception e ) {
                            System.out.println( "Something went wrong - reading thread!!! " + e.getMessage() );
                }

        }
}

/* ******************************************************************************************************** */
```

> ➢ **RESULTS**

Once the server was started, it was visible in TCPView as a listening process. Each connected client was represented by an individual process in established state.

> *a) Networking log from WireShark:*

No. ▾	Time	Source	Destination	Protocol	Info
23 *REF*		155.92.107.81	155.92.106.44	TCP	2849 > 1234 [SYN] Seq=0 Len=0 MSS=1460
24	0.000369	155.92.106.44	155.92.107.81	TCP	1234 > 2849 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
25	0.000385	155.92.107.81	155.92.106.44	TCP	2849 > 1234 [ACK] Seq=1 Ack=1 Win=65535 [TCP CHECKSUM INCORRECT] Len=0
60	7.182545	155.92.107.81	155.92.106.44	TCP	2849 > 1234 [PSH, ACK] Seq=1 Ack=1 Win=65535 [TCP CHECKSUM INCORRECT] Len=13
61	7.371920	155.92.106.44	155.92.107.81	TCP	1234 > 2849 [ACK] Seq=1 Ack=14 Win=65522 Len=0
92	12.226901	155.92.107.81	155.92.106.44	TCP	2849 > 1234 [PSH, ACK] Seq=14 Ack=1 Win=65535 [TCP CHECKSUM INCORRECT] Len=9
93	12.236192	155.92.106.44	155.92.107.81	TCP	1234 > 2849 [PSH, ACK] Seq=1 Ack=23 Win=65513 Len=9
95	12.353227	155.92.107.81	155.92.106.44	TCP	2849 > 1234 [ACK] Seq=23 Ack=10 Win=65526 [TCP CHECKSUM INCORRECT] Len=0
110	15.251546	155.92.107.81	155.92.106.44	TCP	2849 > 1234 [PSH, ACK] Seq=23 Ack=10 Win=65526 [TCP CHECKSUM INCORRECT] Len=9
111	15.259900	155.92.106.44	155.92.107.81	TCP	1234 > 2849 [PSH, ACK] Seq=10 Ack=32 Win=65504 Len=9
112	15.370756	155.92.107.81	155.92.106.44	TCP	2849 > 1234 [ACK] Seq=32 Ack=19 Win=65517 [TCP CHECKSUM INCORRECT] Len=0
162	24.435020	155.92.107.81	155.92.106.44	TCP	2849 > 1234 [PSH, ACK] Seq=32 Ack=19 Win=65517 [TCP CHECKSUM INCORRECT] Len=11
163	24.443375	155.92.106.44	155.92.107.81	TCP	1234 > 2849 [PSH, ACK] Seq=19 Ack=43 Win=65493 Len=11
164	24.624484	155.92.107.81	155.92.106.44	TCP	2849 > 1234 [ACK] Seq=43 Ack=30 Win=65506 [TCP CHECKSUM INCORRECT] Len=0
175	27.379336	155.92.107.81	155.92.106.44	TCP	2849 > 1234 [PSH, ACK] Seq=43 Ack=30 Win=65506 [TCP CHECKSUM INCORRECT] Len=16
176	27.390211	155.92.107.81	155.92.106.44	TCP	2849 > 1234 [FIN, ACK] Seq=59 Ack=30 Win=65506 [TCP CHECKSUM INCORRECT] Len=0
177	27.390501	155.92.106.44	155.92.107.81	TCP	1234 > 2849 [ACK] Seq=30 Ack=60 Win=65477 Len=0
178	27.405046	155.92.106.44	155.92.107.81	TCP	1234 > 2849 [FIN, ACK] Seq=30 Ack=60 Win=65477 Len=0
179	27.405059	155.92.107.81	155.92.106.44	TCP	2849 > 1234 [ACK] Seq=60 Ack=31 Win=65506 [TCP CHECKSUM INCORRECT] Len=0

We can see the three-way handshake here. The client (*.107.81) starts establishing the connection by sending an SYN segment. The server (*.106.44) replies by sending SYN+ACK. The client sends an ACK and the link is established.
There are a couple of segment carrying data in the middle.
The client starts closing the connection by sending FIN. The server replies with ACK and finishes its work. Once it is finished the server sends FIN+ACK. The client sends ACK and that is the end of connection.

> *b) Networking log – server closes its output:*

No. ▾	Time	Source	Destination	Protocol	Info
28 *REF*		155.92.107.81	147.32.107.136	TCP	3222 > 1234 [SYN] Seq=0 Len=0 MSS=1460
76	2.970800	155.92.107.81	147.32.107.136	TCP	3222 > 1234 [SYN] Seq=0 Len=0 MSS=1460
119	8.986589	155.92.107.81	147.32.107.136	TCP	3222 > 1234 [SYN] Seq=0 Len=0 MSS=1460
185	20.702658	147.32.107.136	155.92.107.81	TCP	1234 > 3222 [SYN, ACK] Seq=0 Ack=1 Win=64000 Len=0 MSS=1260
186	20.702706	155.92.107.81	147.32.107.136	TCP	3222 > 1234 [ACK] Seq=1 Ack=1 Win=65535 [TCP CHECKSUM INCORRECT] Len=0
193	22.752865	147.32.107.136	155.92.107.81	TCP	[TCP Dup ACK 185#1] 1234 > 3222 [ACK] Seq=1 Ack=1 Win=64000 Len=0
198	23.875271	147.32.107.136	155.92.107.81	TCP	[TCP Dup ACK 185#2] 1234 > 3222 [ACK] Seq=1 Ack=1 Win=64000 Len=0
206	25.520708	155.92.107.81	147.32.107.136	TCP	3222 > 1234 [PSH, ACK] Seq=1 Ack=1 Win=65535 [TCP CHECKSUM INCORRECT] Len=9
209	25.693246	147.32.107.136	155.92.107.81	TCP	1234 > 3222 [PSH, ACK] Seq=1 Ack=10 Win=64000 Len=9
210	25.830794	155.92.107.81	147.32.107.136	TCP	3222 > 1234 [ACK] Seq=10 Ack=10 Win=65526 [TCP CHECKSUM INCORRECT] Len=0
227	29.630470	155.92.107.81	147.32.107.136	TCP	3222 > 1234 [PSH, ACK] Seq=10 Ack=10 Win=65526 [TCP CHECKSUM INCORRECT] Len=18
228	29.784138	147.32.107.136	155.92.107.81	TCP	1234 > 3222 [FIN, ACK] Seq=10 Ack=28 Win=64000 Len=0
229	29.784189	155.92.107.81	147.32.107.136	TCP	3222 > 1234 [ACK] Seq=28 Ack=11 Win=65526 [TCP CHECKSUM INCORRECT] Len=0
238	31.410095	155.92.107.81	147.32.107.136	TCP	3222 > 1234 [PSH, ACK] Seq=28 Ack=11 Win=65526 [TCP CHECKSUM INCORRECT] Len=7
240	31.717211	147.32.107.136	155.92.107.81	TCP	1234 > 3222 [ACK] Seq=11 Ack=35 Win=64000 Len=0
251	34.283334	155.92.107.81	147.32.107.136	TCP	3222 > 1234 [PSH, ACK] Seq=35 Ack=11 Win=65526 [TCP CHECKSUM INCORRECT] Len=16
252	34.283672	155.92.107.81	147.32.107.136	TCP	3222 > 1234 [FIN, ACK] Seq=51 Ack=11 Win=65526 [TCP CHECKSUM INCORRECT] Len=0
253	34.407128	147.32.107.136	155.92.107.81	TCP	1234 > 3222 [ACK] Seq=11 Ack=52 Win=64000 Len=0

After receiving segment No. 227, the server (147.32.107.136) closes his output, i.e. it sends FIN signal to the client (155.92.107.81). The client acknowledges it by an ACK and knows that no other data will come from the server.
When the client is finished, it sends FIN signal and the server replies by an ACK, that is the end of the connection.
(The multiple SYN segments at the beginning were caused by a firewall.)

> *c) Networking log – client closes its input:*

No. ▾	Time	Source	Destination	Protocol	Info
34 *REF*		155.92.107.81	147.32.107.136	TCP	3279 > 1234 [SYN] Seq=0 Len=0 MSS=1460
45	2.950634	155.92.107.81	147.32.107.136	TCP	3279 > 1234 [SYN] Seq=0 Len=0 MSS=1460
52	3.917431	147.32.107.136	155.92.107.81	TCP	1234 > 3279 [SYN, ACK] Seq=0 Ack=1 Win=64000 Len=0 MSS=1260
53	3.917479	155.92.107.81	147.32.107.136	TCP	3279 > 1234 [ACK] Seq=1 Ack=1 Win=65535 [TCP CHECKSUM INCORRECT] Len=0
57	4.556846	147.32.107.136	155.92.107.81	TCP	[TCP Dup ACK 52#1] 1234 > 3279 [ACK] Seq=1 Ack=1 Win=64000 Len=0
94	12.255790	155.92.107.81	147.32.107.136	TCP	3279 > 1234 [PSH, ACK] Seq=1 Ack=1 Win=65535 [TCP CHECKSUM INCORRECT] Len=9
95	12.417879	147.32.107.136	155.92.107.81	TCP	1234 > 3279 [PSH, ACK] Seq=1 Ack=10 Win=64000 Len=9
97	12.575894	155.92.107.81	147.32.107.136	TCP	3279 > 1234 [ACK] Seq=10 Ack=10 Win=65526 [TCP CHECKSUM INCORRECT] Len=0
146	23.549126	155.92.107.81	147.32.107.136	TCP	3279 > 1234 [PSH, ACK] Seq=10 Ack=10 Win=65526 [TCP CHECKSUM INCORRECT] Len=12
148	23.857971	147.32.107.136	155.92.107.81	TCP	1234 > 3279 [ACK] Seq=10 Ack=22 Win=64000 Len=0
210	32.240800	155.92.107.81	147.32.107.136	TCP	3279 > 1234 [PSH, ACK] Seq=22 Ack=10 Win=65526 [TCP CHECKSUM INCORRECT] Len=9
214	32.610041	147.32.107.136	155.92.107.81	TCP	1234 > 3279 [ACK] Seq=10 Ack=31 Win=64000 Len=0
240	39.074069	155.92.107.81	147.32.107.136	TCP	3279 > 1234 [PSH, ACK] Seq=31 Ack=10 Win=65526 [TCP CHECKSUM INCORRECT] Len=16
241	39.094023	155.92.107.81	147.32.107.136	TCP	3279 > 1234 [FIN, ACK] Seq=47 Ack=10 Win=65526 [TCP CHECKSUM INCORRECT] Len=0
243	39.224987	147.32.107.136	155.92.107.81	TCP	1234 > 3279 [ACK] Seq=10 Ack=48 Win=64000 Len=0
244	39.324735	147.32.107.136	155.92.107.81	TCP	1234 > 3279 [FIN, ACK] Seq=10 Ack=48 Win=64000 Len=0
245	39.324769	155.92.107.81	147.32.107.136	TCP	3279 > 1234 [ACK] Seq=48 Ack=11 Win=65526 [TCP CHECKSUM INCORRECT] Len=0

The client (155.92.107.81) closes its input after sending segment No. 146. The communication continues normally without any change.

True networking logs from WireShark instead of screenshots are available upon request.

> ## CONCLUSIONS

Using the three-way handshake is a bit complicated, but it is a robust method to establish a connection.

When there are more clients connected, they are all connected to the same port on the server. Which thread will ultimately read the incoming data is probably determined by comparison of source address and port.

If one side closes its output, the other side is informed by a TCP segment with FIN flag set. The link is then in the half-closed mode when data can be send in one (the open one) direction only.

On the other hand if one side closes its input, nothing is done on the TCP level.