|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Pratik Pujari** | | |
| **UID no.** | **2020300054** | **Class:** | **Comps C Batch** |
| **Experiment No.** | 7 | | |

|  |  |
| --- | --- |
| **AIM:** | To implement Socket Programming |
| **THEORY:** | **Java Socket Programming**  Java Socket programming is used for communication between the applications running on different JRE.  Java Socket programming can be connection-oriented or connection-less.  Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.  The client in socket programming must know two information:   1. IP Address of Server, and 2. Port number.   Here, we are going to make one-way client and server communication. In this application, client sends a message to the server, server reads the message and prints it. Here, two classes are being used: Socket and ServerSocket. The Socket class is used to communicate client and server. Through this class, we can read and write message. The ServerSocket class is used at server-side. The accept() method of ServerSocket class blocks the console until the client is connected. After the successful connection of client, it returns the instance of Socket at server-side.  Socket Programming in Java  **Socket class**  A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.  **Important methods**    **ServerSocket class**  The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.     * **Socket creation:**   int sockfd = socket(domain, type, protocol)  sockfd: socket descriptor, an integer (like a file-handle) domain: integer, specifies communication domain. We use AF\_ LOCAL as defined in the POSIX standard for communication between processes on the same host. For communicating between processes on different hosts connected by IPV4, we use AF\_INET and AF\_I NET 6 for processes connected by IPV6. type: communication type  SOCK\_STREAM: TCP(reliable, connection oriented) SOCK\_DGRAM: UDP(unreliable, connectionless) protocol: Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man protocols for more details)   * **Setsockopt:**   int setsockopt(int sockfd, int level, int optname,  const void \*optval, socklen\_t optlen);  This helps in manipulating options for the socket referred by the file descriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents error such as: “address already in use”.   * **Bind:**   int bind(int sockfd, const struct sockaddr \*addr,  socklen\_t addrlen);  After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR\_ANY to specify the IP address.   * **Listen:**   int listen(int sockfd, int backlog);  It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.   * **Accept:**   int new\_socket= accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);  It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket.  At this point, connection is established between client and server, and they are ready to transfer data.  **Stages for Client**   * **Socket connection:** Exactly same as that of server’s socket creation * **Connect:**   int connect(int sockfd, const struct sockaddr \*addr,  socklen\_t addrlen);  The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr.  Server’s address and port is specified in addr. |
| **Server Client** | |
| **Code:** | import java.net.\*;  import java.io.\*;  public class chatserver {  public static void deleteString(String st){    for(int i=0;i<=st.length();i++)  System.out.print('\b');    }  public static void main(String args[]) throws Exception {  ServerSocket ss=new ServerSocket(2000);  Socket sk=ss.accept();  BufferedReader cin=new BufferedReader(new InputStreamReader(sk.getInputStream()));  PrintStream cout=new PrintStream(sk.getOutputStream());  BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in));  String s,message;  message="Awaiting for clients's reply!";  System.out.println("CHAT SERVER started ");  System.out.println("Connected to Port : 2000");  System.out.println("Type End to leave Chat");  while ( true )  {  System.out.print(message);  s=cin.readLine();  if (s.equalsIgnoreCase("END"))  {  cout.println("BYE");  break;  }  deleteString(message);  System. out.print("\nClient : "+s+"\n");  System.out.print("Server : ");  s=stdin.readLine();  cout.println(s);  }  ss.close();  sk.close();  cin.close();  cout.close();  stdin.close();  }  } |

|  |  |
| --- | --- |
| **Client Server** | |
| **CALCULATION:** | import java.net.\*;  import java.io.\*;  public class chatclient {  public static void deleteString(String st){    for(int i=0;i<=st.length();i++)  System.out.print('\b');    }  public static void main(String args[]) throws Exception  {  Socket sk=new Socket("127.0.0.1",2000);  BufferedReader sin=new BufferedReader(new InputStreamReader(sk.getInputStream()));  PrintStream sout=new PrintStream(sk.getOutputStream());  BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in));  String s,message;  message="Awaiting for server's reply!";  System.out.println("CHAT CLIENT Connected");  System.out.println("Connected: 2000");  System.out.println("Type End to exit the chat");  while ( true )  {  System.out.print("Client : ");  s=stdin.readLine();  sout.println(s);  System.out.print(message);  s=sin.readLine();  deleteString(message);  System.out.print("\nServer : "+s+"\n");  if ( s.equalsIgnoreCase("END") )  break;  }  sk.close();  sin.close();  sout.close();  stdin.close();  }  } |

|  |  |
| --- | --- |
| **Screenshots** | |
| **OUTPUT:** | Firstly, start the chatserver.java in order start a socket      Then initialize the chat client for the sock of port 2000 to make a request an accept it  After a connection is made server side should look like this      Client side    Now simply type anything whenever it’s your chance          The chat ends with client says End or the server side says Bye |
| **RESULT:** Learnt about the socket programming and its features. Learnt on how to generate a socket using the Socket class. Used the function/methods of BufferedReader and PrintStream in order to get and send messages of users. | |