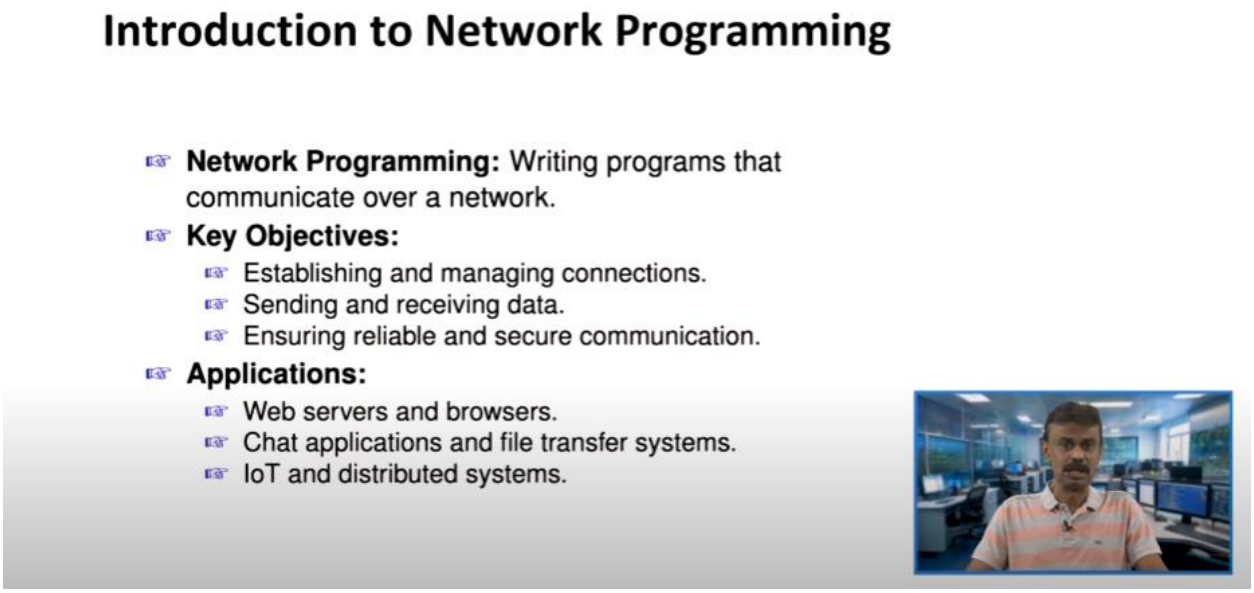
FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture53

Lecture 53: Introduction to Network Programming



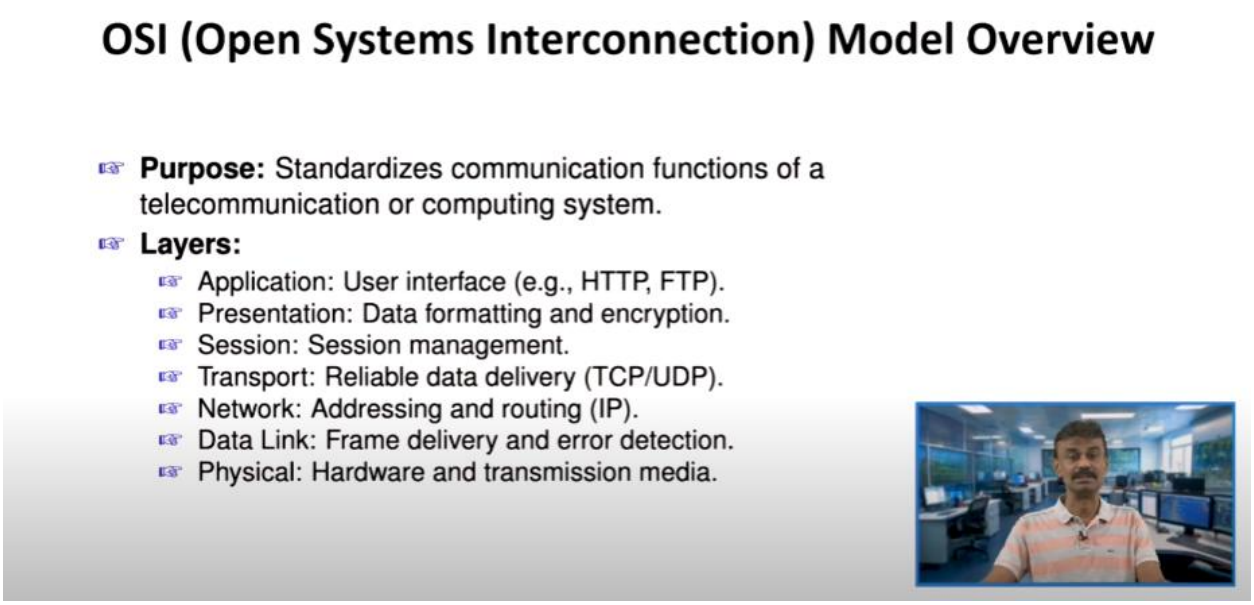
Welcome to lecture number 53, Advanced Topics in Object-Oriented Programming. So in this lecture, I'll start with an introduction to network programming, right? So, what do you mean by network programming? Suppose you are writing a program that is communicating with more than one device, right? So, two or more devices, precisely, right?

So, you are writing a program so that we can communicate with several devices. In fact, nowadays, it is happening, right? So, we have several chat devices. Devices or chat applications, or even the web servers, right? So, we can communicate. The main objective is to establish and manage the connections, right? How to establish the connection, how we are going to manage the connection—that is the first objective—and then sending and receiving the data, right?

So, you are sending the data, and the other person or even the group can receive the data. All right. Or they are sending the data. You are receiving the data. Right.

So, sending and receiving data is the next objective. And the third one is ensuring reliable and secure communication. So, whether you are getting the message from a reliable source and also when you are sending, how secure your message is. Right. So, these are all the main key objectives in network programming and day-to-day life.

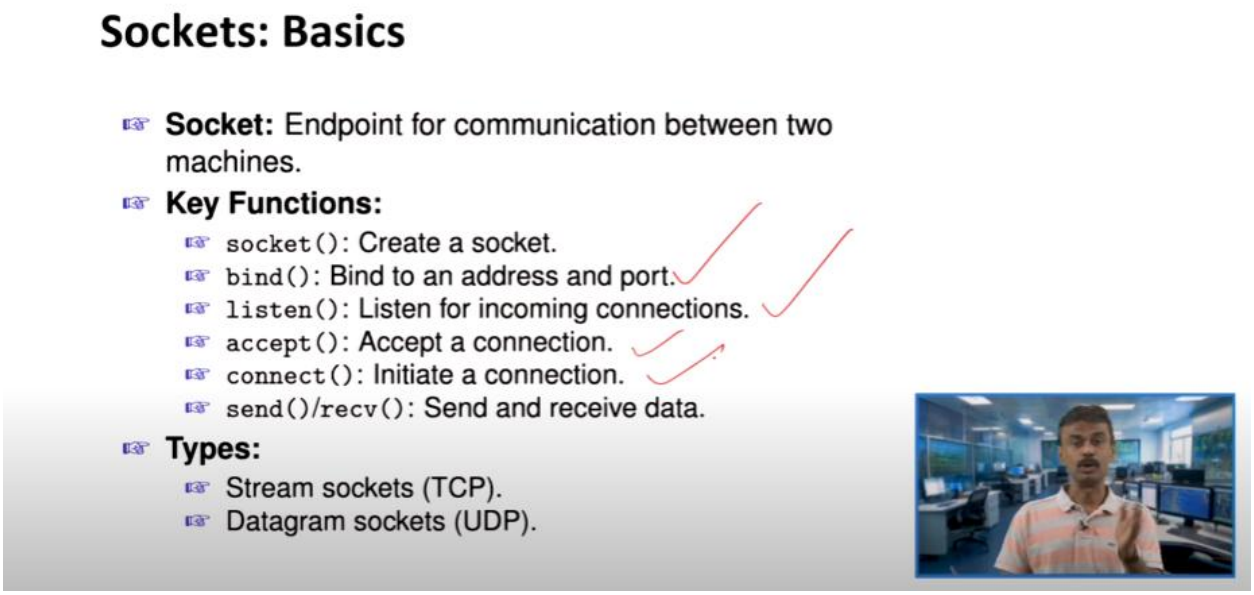
We have several applications. Everyone is using web browsers, web servers, and chat applications. So, nowadays, it is very common. And even if you want to transfer a file, I mean, you can do that. Even making a video call, fine, right.



So, you can do that. So, these are all the important applications. And other than this, the IoT and distributed systems, Internet of Things and distributed systems, we have several applications, right. So, this is the basic introduction to network programming. So now, what do you mean by the Open Systems Interconnection model, the OSI model, right?

This is one of the standard questions, standard interview goes in. What are all the layers in OSI, right? So we have seven layers. The seven layers go like this, application layer, presentation layer, session layer, transport layer, network layer, data link layer, and physical layer, right? So the main purpose of OSI

to standardize the communication functions of a telecommunication or computing system for example if I take application layer the user interface you have HTTP and FTP right file transfer protocol and the presentation presentation layer presentation layer so when you want to have an encryption Or when you want to have a data formatting. Right. So we use the presentation layer.



And then session layer is for session management. Right. And the transport layer. The reliable data delivery. The TCP UDP.

The reliable data delivery. And addressing and routing. You have a network layer. Right. The IP address, etc.

Network layer. So you call it addressing. IP address. Internet Protocol address. Data link layer.

So. The frame delivery and error detection, right? If you want to find the error or if the error is detected or the delivery of the frame. For this, the data link layer is helpful. And hardware and transmission media, physical layer. So, these are all very important, and in fact, the seven layers, right.

So, in fact, it is a very common interview course in what are all the, I mean, seven layers that you know in OSI, right, OSI model, open system. interconnection model. So, first what do you mean by sockets? So, sockets which are nothing but the endpoint communication. So, endpoint for communication between two machines.

So, when you are having a two machines. So, the endpoint for communication you call it as a socket. So, the key functions let us say suppose I use socket that will create a socket. Bind we are going to use bind. Bind to an address and port.

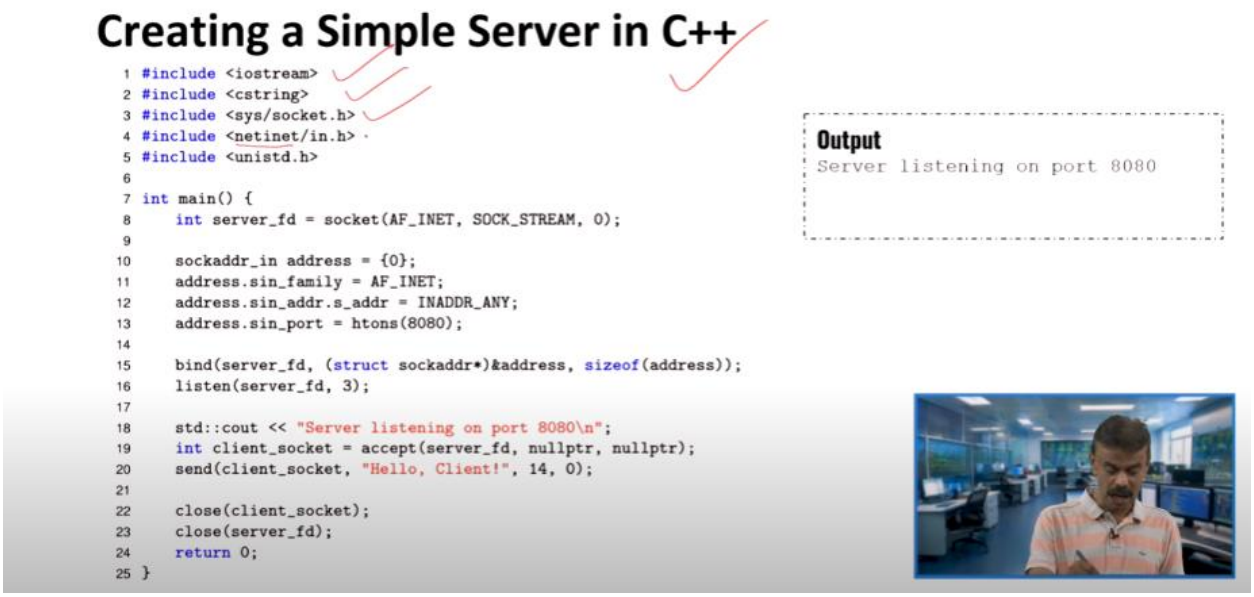
Bind to an address and port. Another function is called listen. right listen for incoming connections accept a connection initiate a connection connect right you have accept accept a connection listen listen for incoming connections, connect function is for initiate a connection, send or receive right you are sending and receiving a data right so to send and receiving a data you have send or receive function right so these are all the key functions

And what are all the different types of sockets? So, you have TCP, right? Which is for stream sockets, right? Stream sockets are for TCP. And datagram sockets are for UDP, right?

So, these are some of the basics of sockets. So, let us consider, right? This first program in network programming. Creating a simple server in C++, right? So, how are we going to create a simple server in C++?

Let us have iostream and cstring. So, here you have sys/socket.h, right. So, this contains socket-related functions such as socket, bind, and accept, as we have just seen, right. So, this header file contains socket, bind, and accept. Another one is include netinet, right.

So, the net inet, you have socket address underscore in or it defines the structures and constants for internet domain addresses, right. So, the internet domain addresses. So, which has a structures and constants. The last one is unistd, right. So, this provides close function to close file descriptors.



So, this is useful for close function right. In fact, it is providing the close function right. So, which is for closing the file descriptors. So, with this we will go to the main program. So, the main program you have an object server.

In fact, it is a variable server underscore f integer variable right. So, the right hand side you have the socket function. You have AF underscore INET. So, this is indicating the use of IPv4. internet protocol version 4 right and then you have SOCK underscore stream it is specifying TCP socket right.

So this is specifying TCP socket and then the 0. So this uses the default protocol for TCP. So 0 is for the default protocol for TCP right and then So whatever it is returning that socket is returning it will be stored in server underscore fd which I already said which is the integer variable right. So now line number 10 you have SOCK address underscore in right.

So which is the object under this object you have address. So that address is initialized to 0 so that means this is initializing the address structure which define the server's address and port right. So, this is defining the server address and port right. So, the next one is address is invoking the data member right sin underscore family right.

So, which is af underscore inet. So, this specifies the address family as IPv4, right. So, this specifies the address family as IPv4. So, the next line you have address is invoking SIN underscore address which is invoking S underscore address, it is address, right.

So, the right hand side you have in address underscore any, right. So, what do you mean by this? So, this binds, right. So, this binds the socket to any IP address on the host machine.

So, that is the meaning, right? So, this binds the socket to any IP address on the host machine. And line number 13, you have address SIN underscore port, right? HTONS of 8080, right? So, this sets the port number to 8080, right?

Sets the port number to 8080, right? And converting it to network byte order using edge tones. That means host to network short. The meaning of edge tones is nothing but host to network short. short all right.

So, this is happening over here. So, next come to the next line that means line number 15 bind right. So, here you have three parameters one is server underscore FD right server underscore FD and then the address correct. So, here you have the reference of the address under the structure pointer SOC address. and then size of address right.

So, this binds a server socket right. So, this binds a server socket server underscore fd to the address and port specified in address right. So, this is what this particular function is doing right. So, you are passing server underscore fd and reference of the address under the structure point of SOC address and the size of address. So, that means this binds

That is server underscore fd right to the address and port specified in address that is what line number 15 is doing. Then line number 16 you have listen you have two parameters server underscore fd comma 3. That means this puts the server in listening mode allowing it to accept to three pending client connections right. So, three pending client connections right. So, three pending client connections.

So, next one is a cout statement. So, that means this will be printed server listening on port 8080 will be printed right. So, it is a simple print message and the next line client socket is a variable integer variable and in the right hand side you have accept function this we had seen right accept bind etcetera accept function. So, server underscore fd null pointer null pointer. Right.

So here what it is happening? It is waiting for a client to connect and returns a new socket file descriptor. Right. Returns a new socket file descriptor, which is called as a client underscore socket for the specific client connection. For the specific client connection.

And then null pointer is passed for the client address and length client address and length. right, as they are not required here, right. So, this is up to your line number 19. So, now, let us be in line number 20. So, line number 20, you have send, send function, all right.

We are calling the send function. So, passing three parameters, client, socket, hello client, 14, in fact, four parameters and 0, right. So, it sends the string hello client to the connected client. right. So, 14 stands for length of the string, 14 stands for the length of the string and 0.

So, you can say that there is no special flags are used, no special flags are used, right. So, this is whatever send function then you have close client socket. So, that closes the connection with the client and line number 23 you have close server FD. So, that closes the server socket and then you have return 0 right.

So, when you run the code right. So, here you are getting the output server listening on port 8080 will be printed right. So, this is what happening over here. So, this is how you can create. So, this is output we are getting.

So, that means this is what right we are creating a simple server in C++ right. So, this is what these are all the functions we use. So, first is for creating a socket that I explained all right. So, which is for IPv4 and TCP correct. So, this is what we have done in line number 8 right.

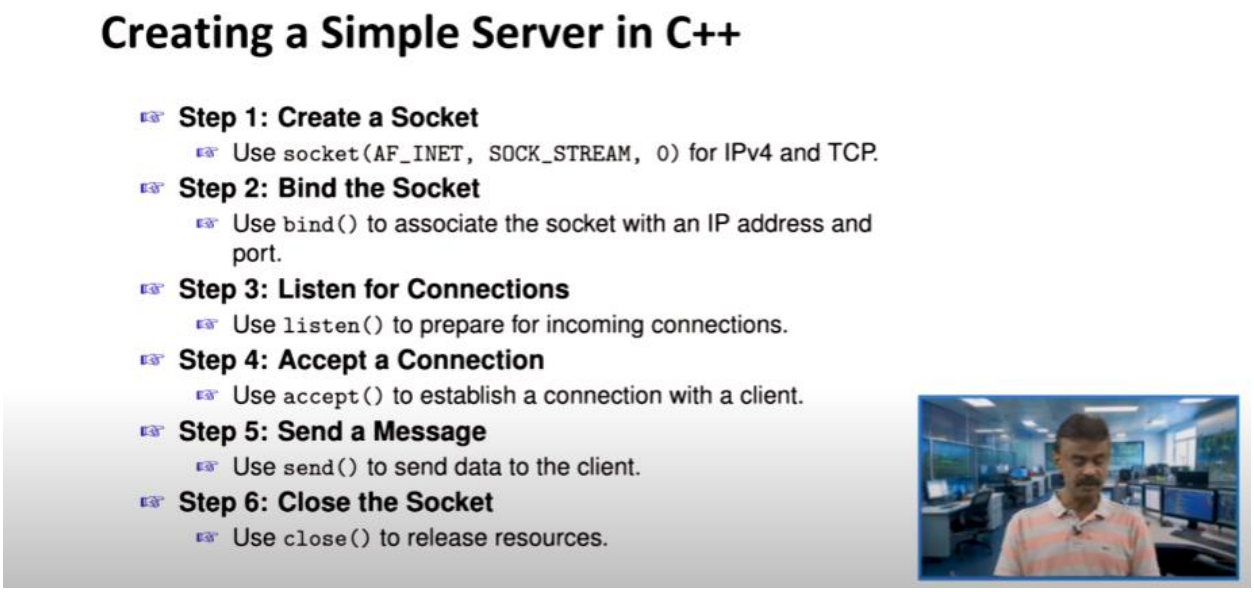
So, that is for right. So, creating right socket yeah socket is in the right hand side. And bind the socket we use the bind function and listen to connections I explain right listen. So, which is available in line number 16 bind is available in line number 15 right. So, then accept a connection use accept right.

So, accept a connection we have used in line number 19 this also we have seen with the 3 parameter server underscore FD null pointer and null pointer and send a message right. So, we used to send a message using. the send function so which is available in line number 20 right so and then close the socket right so this is how you can create a simple server in c++ so the next program we will see creating a simple client in c++ right so here we have included iostream right so iostream and cstring like a last program and here we include sys slash socket dot h

Right, the third header file. So, this provides socket-related functions like socket, connect, and read, right? And the fourth one is arpa/inet. So, this contains the inet_pton

function to convert IP addresses from text to binary form, right? IP addresses from text to binary form. So, the inet_pton function is available in arpa/inet.

And the next one is So, which provides the close function to close the socket, as we have already seen in the last program. So, now look at line number 8. We are coming to the main line number 8. You have client_fd, which is an integer and, like in the last program, socket AF_INET (for IPv4) and SOCK_STREAM, which indicates a TCP connection. Right, and 0, which represents the default protocol for TCP.



So, on the left-hand side, you have client_fd, which is used to refer to the socket. Now, line number 10, you have sockaddr_in, and here you have server_address, which is equal to {0}. So, this represents That means it is initializing the server_address, right? The server_address structure, right?

The server underscore structure which stores the server's IP address and port, right? So this is storing server's IP and, right? IP address and port. So the next line you have server underscore address, right? dot SIN family which is equal to AF underscore INET.

I already told this is specifying the address family as IPv4. So, the next line you have server underscore address dot SIN underscore port which is HTONS8080. So, this we have already I mean talked about this right. So, this is setting the server port to 8080. and converts into network byte order using H tones right.

So, this we had already seen and next one INET underscore pton right. So, you have three parameters one is AF underscore INET. So, that means you are right three parameters right now I am saying three parameters 1, 2 and 3 right. So, what exactly is happening? So, you have

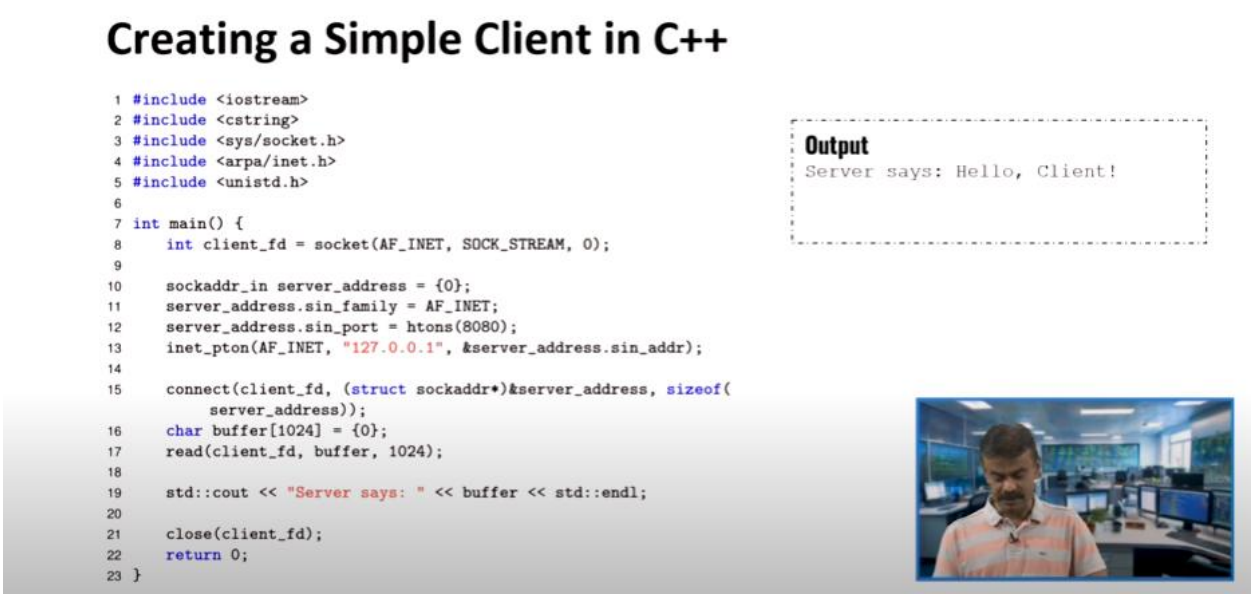
The loopback IP address. So this is called the loopback IP address. Right. So that is 127.0.0.1, which you call it as a local host. Right.

So from text to binary. Right. So this is converting the loopback IP address. Right. 127.0.0.1 from text to binary and stores it in server.

Right. So, this binary form is being stored in the server, right and in fact it is storing it in server underscore address dot sin underscore address. So, here it is being stored right. So, converting and then it is being stored over here right. So, the function right INET underscore pton

So, which is ensuring proper address formatting either it will be IPv4 or IPv6 right. So, this is ensuring the proper address formatting INET underscore pton is ensuring the proper address format okay. So, either it will be IPv4 or IPv6. So, now the next function connect right. So, which we have already seen right in the previous program.

for the server client underscore fd and then this is the address and the size of the address right. And the next line line number 6 in this we have already seen the last hour I have explained in the case of simple right creating a simple server. So, that will hold here also. So, line number 16 you have buffer of 1024 right buffer of 1024 you are initializing to 0. So, the meaning is this is declaring a buffer size of 1024 to store



the data received from the server initializing all the elements to 0 right it is initializing all the elements to 0 right that is the meaning. And the next function you have read client FD buffer and 1024. So that means it is reading up to 1024 bytes of data from the server into the buffer. So here we have client underscore FD. So that means the socket used for communication with the server and you have buffer which stores the data received from the server.

Right. What about client_FD? So, the socket used for communication with the server, right. So, this is up to your read, and then you have a see_out statement server says. So, the buffer is right.

So, that means whatever data is received from the server will be displayed, right. Whatever data is received from the server will be displayed, and then close client_FD, right. So, this is closing. And then you are returning 0. So, when I run the code here, you are seeing right, server says that will be printed, and the buffer you are getting: hello client, right.

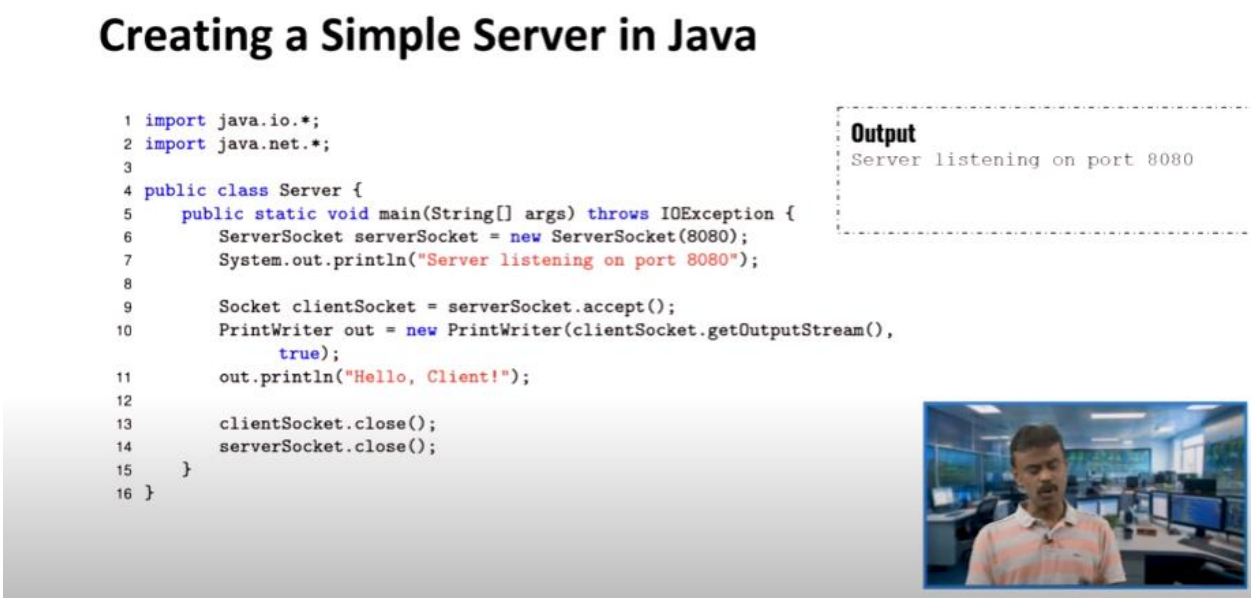
So, that means this is receiving this data, right. So, the data received from the server is hello client, right. So, data received from the server is hello client. So, that will be printed. So, because it is being stored over there, that is being printed, okay.

So this is what the output you are getting. So, these are all right the functions the important functions we had seen right I explained about socket for IPv4 and internet protocol version 4 and TCP and I also talked about the SOCK address underscore in

and we have also seen right the IP address is being converted into binary. So, INET underscore PTON is useful. And we have used the connect function and we read server's message, right?

So yeah, the read function we have used and also for displaying a message. Yeah, that is fine. See out. This is standard one. Close the socket.

We use the close function, right? All these we had seen. So this is how we can able to create simple server and simple client using C++. So, the same code in fact, I will run the code for Java. So, the same whatever we had seen in C++.



So, we can do it in Java also. So, let us create a simple server in Java. So, whatever I talked about over there. which will be right to hold good in java also. So, java dot io utility star and java dot net dot star these utility is being used and you have a public class server which is the driver class and I have the main program.

So, which throws io exception right. So, I have a server socket object right under the class server socket right because I have put all these utilities and in the right hand side you have memory allocation with with constructor by passing 8080 right. So this is a similar code right what we had seen in C++ and you are printing one message server listening to port 8080 and here you have client socket object right under socket right and

in fact the client socket is invoking accept like the accept function that we had seen in C++.

Similar to that, this will work over here. And here I have out, right, object. So, out object is under print writer. So, the constructor, after allocating a memory, dynamic memory, the constructor, we are passing get output stream and the true value, all right. So, here you have a statement out.println hello client.

So then, as usual, the client socket is being closed and the server socket is being closed. So when you run the code, you will get the output: line number 7, 'server listening on port 8080,' will be printed. So whatever I explained in C++, similar to that, this will work in Java as well. So now, what we will do is we will run the code in Java, right? So we will go to the Java compiler, and what we can do is we will run the code. So let us create the simple client. So this is the output we are getting: 'server listening on port 8080,' right? 'Server listening on port 8080,' right?

So when you run the code, we are getting this particular output. So here, you have created 'SimpleServer.java,' right? So when I run the code, I am getting the output. So, we have seen how to create a simple server and a simple client in both C++ and Java. And in the next class, I will start from, right, the HTTP and REST basics. Thank you very much.