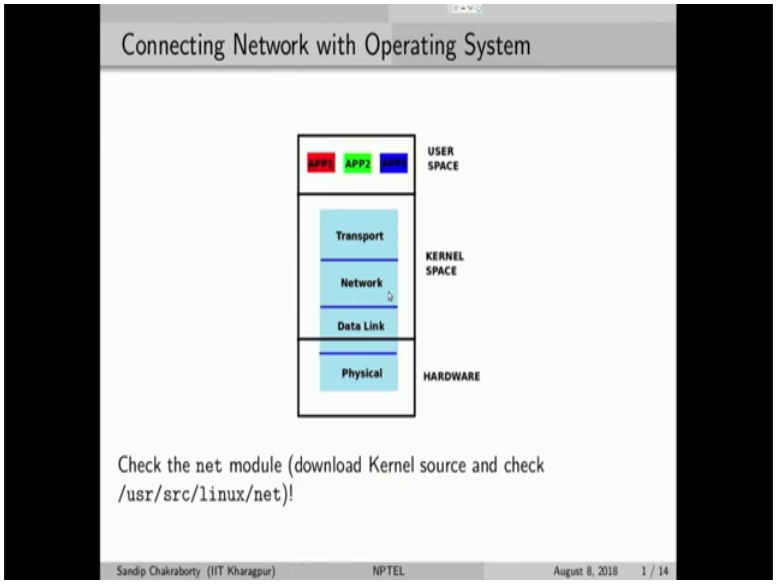
Computer Networks and Internet Protocol
Prof. Sandip Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 24
SOCKET PROGRAMMING – I

Welcome back to the course on computer network and internet protocol. So, today we will see some demo of network programming. So, we will look into the socket programming in details. And see that a with the help of socket programming how you can access the transport layer of the protocol stack, and you can run your own application on top of the network protocol stack to transfer some data to or from a computer.

(Refer Slide Time: 00:47)



So, let us start our journey in the socket programming. So, to start to with as we have seen earlier or we have also discussed earlier that this entire network protocol stack is implemented inside the kernel of the operating system. So, in general we have five layers in the TCP IP protocol stack that we have talked about. And in that five layers of the TCP IP protocol stack this physical layer, and the part of the returning clear are stored inside the hardware.

Whereas the upper part of the data link layer mainly a part of the MAC and dialogical link control the network layer and the transport layer they are implemented inside the

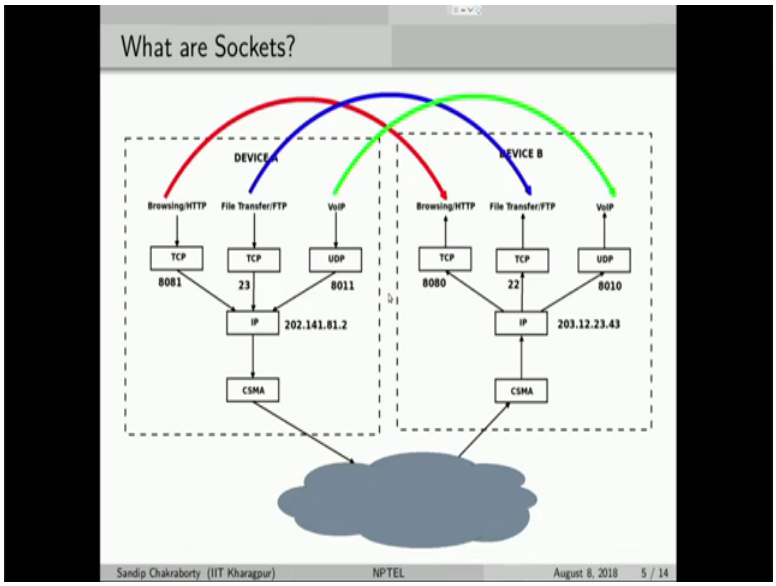
kernel space of operating system. And then from the user space you can write multiple application to access this protocol stack. So, the application have to interact with the transport layer to send or receive data. And as we have learned already that in the transport layer, we have two different protocols; the transmission control protocol or the TCP and the user datagram protocol or UDP.

So, today we will look into that this entire kernel space which is there inside that we have this different layers of the protocol stack, the transport layer, network layer and the part of the data link layer which is already implemented inside the operating system kernel. And we will see that how you can make a interaction between this user space and the protocol stack which is implemented inside the kernel space with the help of the socket programming. Now, interestingly what we will see that whenever you will write a application, network application at the user space, you have to transfer or you have to use certain kind of functionalities which are available at the kernel.

So, you require a interfacing between the user space and the kernel space. And remember that here we are talking about a UNIX based operating system. And, in a UNIX based operating system from the user space to kernel space that interaction will be done with the set of APIs which we call as the system call. So, the system call transfers some user requirement to the kernel corresponding kernel operations and performed operations at the Linux or UNIX kernel. So, this entire protocol stack that is implemented inside the kernel we will make an interaction with that network protocol stack with the help of the operating system, system calls.

So, interestingly just a pointer for you to explore further that you can look into this entire protocol stack implementation inside a UNIX kernel. If you download the UNIX kernel source inside the UNIX kernel source, you can check the net module under user source Linux net. And you can see this entire implementation of the kernel protocol stack there.

(Refer Slide Time: 03:51)



So, these things you have already learnt about that this TCP IP protocol stack at the transport layer it does application layer multiplexing. You can run multiple application in different devices. Now, different application can run different type of protocols. So, if you are having a browsing or http application that will run TCP. If you have a file transfer or FTP kind of application that may again run TCP. If you have some application like VoIP, that may use UDP.

Now, all these different transport layer instances of the protocol stack that interact with the IP layer. Now, whenever you are talking among two devices the IP layer gets changed and whenever you are have multiple applications which are running on top of your IP layer or the network layer of the protocol stack, they are multiplexed with the help of different type of protocol. So, to differentiate between two devices at the IP layer, we use this IP address. In the subsequent lecture, whenever we discuss about IP layer, we will discuss about how you can configure these IP addresses. So, for the time being just understand that different devices, they are separated by the IP addresses. And then at the application side, we have these port numbers that actually help you to do the application layer multiplexing

So that means, so whenever we talk about that device A is communicating with device B during that time it is actually if you are doing some kind of browsing application which is using http protocol during that time the device A, the application which is using the

TCP protocol at a port 8081 on the machine with IP address 202 dot 141 dot 81 dot 2, it is interacting with the device B at IP address 203 dot 12 dot 23 dot 43 over a port 8080, where your http server is running.

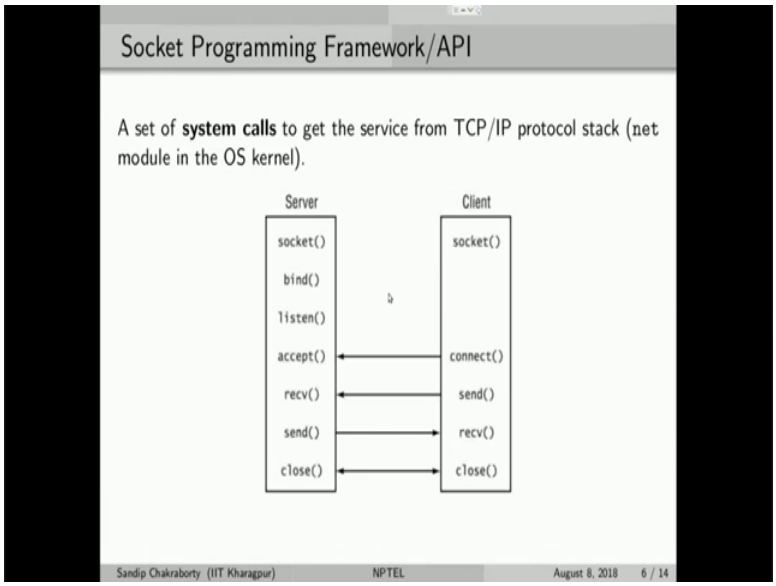
Similarly, for the other application that way this application running at device A on application running a device B they can be segregated with the help of that port number. And remember that whenever we talk about this kind of application in a UNIX based system, we basically represent it in the form of a process. So, there are multiple processes which are running in different machines. And those processes want to communicate with each other and during that time we make or we ensure this process to process communication with the help of that transport layer of the protocol stack.

So, this process to process communication is achieved with the help of this IP address, which is there at the IP layer to uniquely identify a machine in the network. And then in a machine there are multiple processes running, they can use different protocols. Some of the processes may use the TCP protocol, some of the process may use UDP protocol, so that are segregated with the help of the port numbers.

Now, let us see what is a socket. So, socket is basically a logical connection from one process to another process. So, here you can see that these two browsing applications at the two devices they are communicating with each other. So, we have the socket one socket - this red socket which is making a logical pipe between the application which is running at port 8081 on the machine 202 dot 141 81 dot 2 to a machine where the corresponding end of that pipe is running at a port 8080 at a machine with IP address 203 dot 12 dot 23 dot 43. So that way, we can have multiple such logical pipes at the transport layer which we call as the socket.

Now, sending the data over the internet means sending the data over these logical pipes. So, these logical pipes which we call as the socket they basically creates this end to end connection in case of TCP or end to end data transmission semantics in case of UDP to transfer the data from one machine or better to say one process running at one machine to another process running at another machine.

(Refer Slide Time: 08:29)



So, here let us see that how we can implement such a socket in a UNIX based system. So, for that we use this concept of socket programming. So, in a socket programming framework, we have a set of system calls that we can execute from the C program. And this system calls will help us to get the service from the our TCP IP protocol stack which is implemented inside the net module of the OS kernel.

So, let us see that how this entire thing works. So, at the transport layer, we are talking about a client server this programming. So, we have a server where the idea is that the server has opened a port, announce the port data this particular port I am listening and the client need to make a connection to that particular port.

Now, in that case how the server actually works. So, in the server side, you have to first make a socket system call. So, the socket system call, it will create the server side opening of dialogical pipe and it will bind the socket with your TCP IP protocol stack. So, to bind the socket with the TCP IP protocol stack, you have to call this bind function. So, this bind function, what it will do that with the port number that you are specifying it will bind that port number with the socket, so that way it will create a logical end of the connection at the server side.

So, just think of the server in this way that the server is always running, and the server actually need to announce that hey, I am actually listening in this particular port say port 8080. So, if anyone wants to talk to me, you can send data at the port 8080. So, this

announcement you have to done through this bind and a listen system call that we have here. So, the bind system call actually bind the port with the corresponding socket end; and the listen system call will help you just to make the server to go in the listening state. So, the server is now say bind that port 8080, and it is listening for the incoming connection.

Now, let us move at the client side. At the client side you have the socket system call. So, this socket system call again create a client side end of the logical pipe. And after that at a client site you do not require this bind and listen, because just understand the nature of the communication between the server and the client. So, the server is actually announcing or making an announcement that hey, I am listening at this port 8080. So, anyone wants to connect to me, you can directly connect to me at port 8080 that the client do not need to know because the client is actually initiating the connection to the server.

Because the client is initiating the connection to the server, the client do not need to make such kind of announcement. So, the client can just initiate the connection to the port which is being announced by the server and that is why you do not require the bind and the listen call at the client side. So, at the server side, you require the bind and the listen call so that the server can bind itself to a port to a fixed port and it can announce that fixed port to the outside that anyone can connect to that particular port by creating a socket.

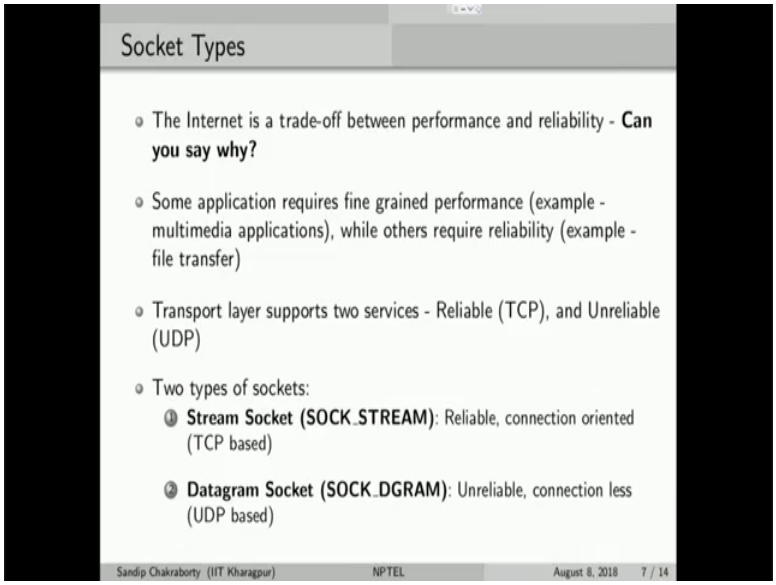
Now, after these things are done, after you have created the end of the socket at the client side from the client side you make this connect call to initiate a connection to the port number which is announced by the server. Now, that is actually a kind of well known things, like say you know that if you are running a http server, then you are either running at port 80 or you are running at port 8080 or some other ports which is being announced by the server. So, the client already knows that what is the IP address where the server is running, and what is the port number where the server is running. So, the client initiate a connect call there and this connect makes a connection towards the server. So, once the server gets this connection, it makes a accept call.

Now, in case of a TCP kind of protocol within that connect and accept, you have the TCP three way handshaking procedure that we have discussed. So, the client initiates the

connection by sending a syn packet, the server accept the connection by returning back an ack; and also initiating the connection to the client side by sending another syn. So, we are having a syn plus ack from the server to the client, and then finally, the client sends an acknowledgement, so that way through this three way handshaking of TCP which happens whenever you are making this connect system call at the client side and accept call at the server side to make the connection in case of a TCP.

Now, once this connection is established, then you can make this send and receive call to send the data and receive the data. So, whenever you are making a send call, it is sending the data; at the other end you can receive that data by making a receive system call or you can make a send to the from the server side to send some data from the server to the client. And the client accept that data from this receive system call. So, once this data communication is done, then you make this close call finally, the close call to close the corresponding connection, so that way this entire flow of socket programming works. Now, let us look into that how you will actually write this system call in the format of a C syntax.

(Refer Slide Time: 14:07)



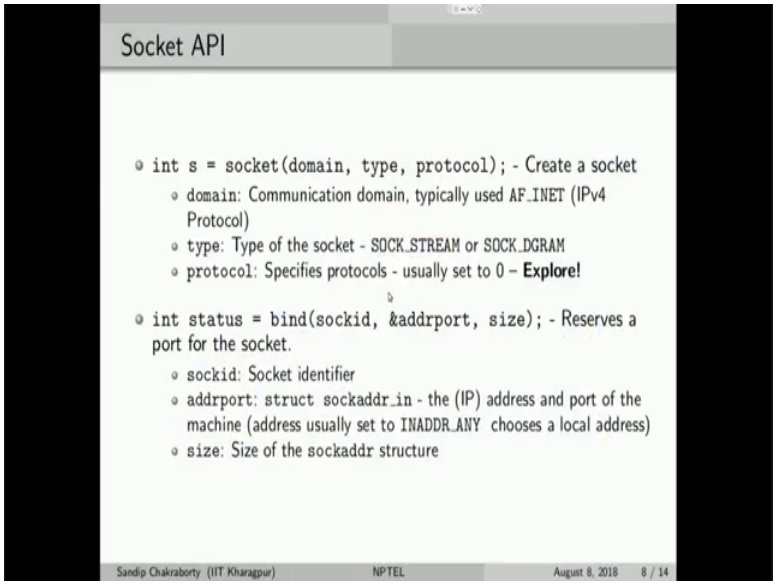
So, we will start with different type of sockets as we have discussed long back that the internet is a tradeoff between performance and reliability and that is why we have two different protocols at the transport layer, the transmission control protocol or TCP or the user datagram protocol and the UDP. Now, some application they require fine grained

performance like the multimedia applications and some others requires reliability like a file transfer. And accordingly we have two services like a reliable transmission protocol or TCP kind of protocol and the unreliable transmission protocol like a which is UDP protocol.

Now, accordingly we have these two different types of sockets; one socket we call as the stream socket which is initiated by sock stream. So, this sock stream is, create a socket which is reliable and connection oriented. So, it is necessarily a TCP kind of socket. On the other hand, we have this UDP based socket which is unreliable and connectionless that we call as a datagram socket, which is termed as sock DGRAM, so that way we have two broad kind of socket stream socket under datagram socket.

Apart from that we have a third kind of socket that is called raw socket using that raw socket you can actually bypass the transport layer and you can directly in turn interact with the IP layer. So, we will not going to discuss this raw socket here in details. We are going to give you an overview about this stream socket and the datagram socket.

(Refer Slide Time: 15:35)



Now, whenever you are declaring a socket, so what you can do you can be clear a very two variable called integer s integer type of variable which hold the socket id that you are going to define. So, this socket system call it takes these parameters, three parameters - the domain, type and the protocol. Now, it creates a socket with this socket system call. Now, this domain parameter it is the communication domain. Normally, we use IPv 4

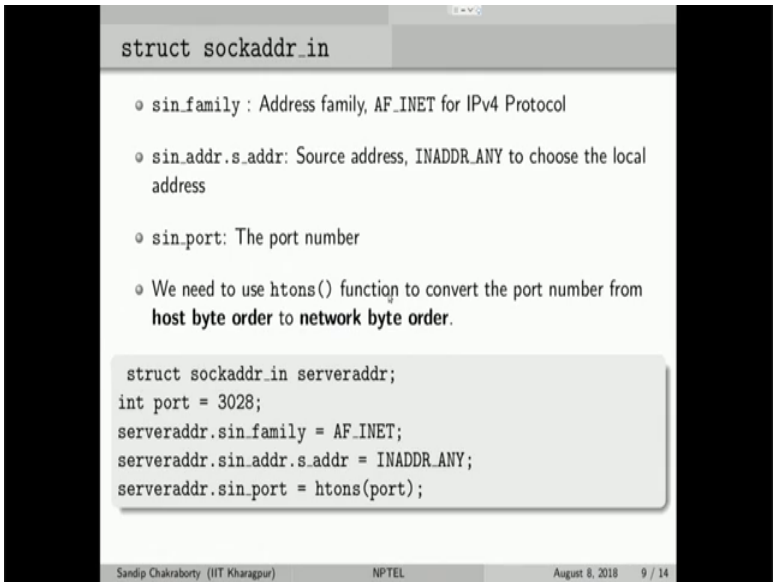
protocol or IPv 4 address. So, we set this domain value as AF underscore INET which is a standard for the time being most of the time you will use a AF INET. You can always explore what are the other possibilities in this domain field.

Then the type of the field it is type of the socket either soft stream or soft datagram based on whether you are going to create a TCP socket or a UDP socket. And finally, the protocol specifies the protocol family that we are going to use usually it is set to 0. So, I will suggest you to explore this that why we set the protocol field at 0 in most of the cases.

Now, once the socket system call is done, you have created the socket. At the server site you have to create the next calls to bind to bind the port to the particular socket. So, this bind system call works in this way it returns the status whether the bind is successful or not. So, you can have the status as integer variable. And the bind takes three parameters the socket id. The socket id that is written by this socket system call and yeah the socket id that is written by this socket system call, so this s value is the socket id which has been written that you can put here.

Now, that particular socket is bind to a address port kind of variable which is a structure. So, this structure contains sock sockaddr in. So, this structure contains the IP address and the port of the machine. So, usually set to inaddr any to choose a local address. So, if you are run it as inaddr any, it will choose the IP address which is used by your machine and then the size. So, the size is the address size of thus this sockaddr structure.

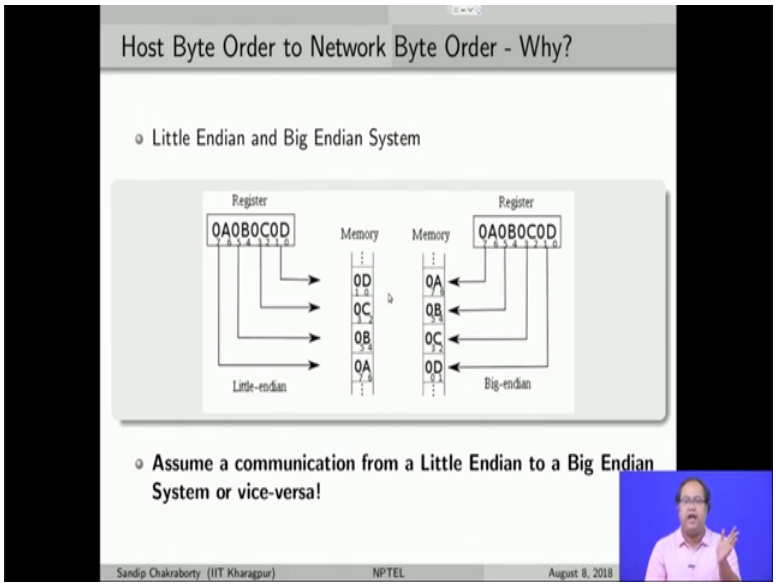
(Refer Slide Time: 18:03)



So, the sockaddr structure looks something like this, which actually stores the IP address and the corresponding port number. It has districts fields one is the sin family. So, this is the address family. So, the address family we keep it as AF INET for IPv4 protocol which we are going to use. So, this concept of IPv4 we are going to discuss in the subsequent lecture. So, when we talk about the IP addressing scheme, so normally in general in today's network we mostly used IPv4 address. So, the address families normally set to AF INET. .

Then we have this socket in address dot s addr it is the source address. So, the source address we keep it as inaddr any as I have mentioned to choose the local address of the machine where I am running the code. And then I have the port number in the variables sin port the port number. So, now one interesting fact is here that we need to use this function called h to ns to convert the port number from host byte order to network byte order. Now, let us look at quickly that what is the source byte order and the network byte order.

(Refer Slide Time: 19:25)

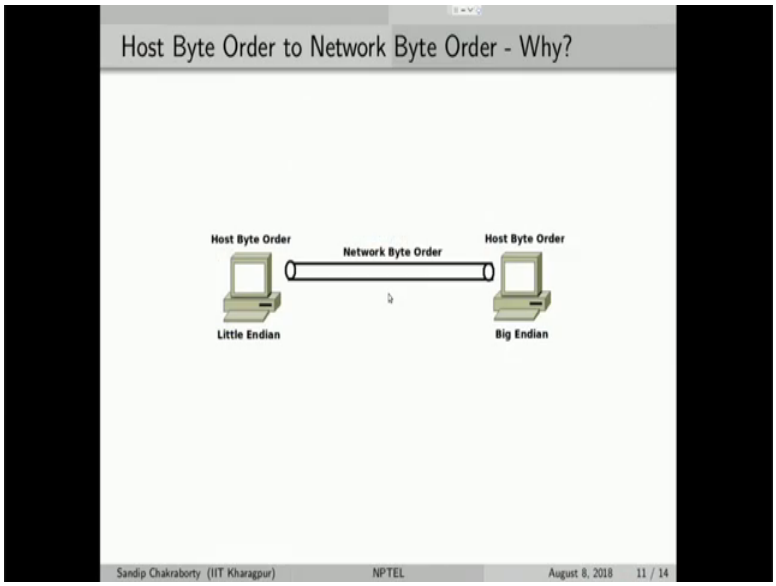


So, in a computer system, the computer can be of two type either it can be a little endian system or it can be a big endian system. Now, the difference between a little endian system or a big endian system is something like this. Yeah, it is like how you are storing the data in the memory. Now, in case of a little endian system, you will store the data from left to right sorry from right to left. So, this 0D will be stored fast then it will store 0C, then it will store 0B, and finally, it will store 0A.

Whereas, in a big endian system, it is just opposite. So, it is left to right associatively kind of things. So, in a register if your data is something like this, in the memory it will first store 0A, then it will store 0B, then it will store 0C and finally, it will store 0D. So, that way depending on whether your machine is following a little endian platform or a big endian platform, the representation of the data inside memory may get changed.

So, assume a communication from a little endian to a big endian system. Now, if you are transferring data from a little endian system, you will transfer the data in the form of a byte stream in the sequence of bytes. So, you will first say and possibly 0D, then 0C, then 0B, then 0A. And whenever it big endian system will get 0D, it will put the 0D first, then the 0C second and that way whenever it will interpret it will interpret the number just in the opposite direction. So, that way there may be a kind of inconsistency whenever you are transferring the system, so that is why we use this concept of post byte order to the network byte order, the idea is that the host can be little endian or big endian.

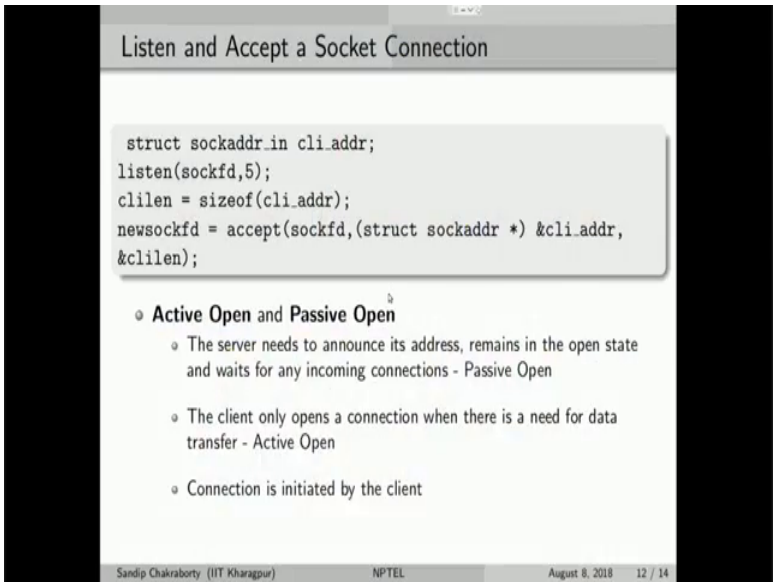
(Refer Slide Time: 20:57)



They have a kind of byte order. Now, network is a has a fixed byte order. So, whenever you are transferring the data over the network, you convert it from the host byte order to the network byte order, transfer it over the network. At the other end, you fetch the data convert it again to the host byte order based on whether the your system is little endian or big endian and store it there. So that way this kind of inconsistency which may come due to the representation difference of two machines that can be solved, so that is the idea of converting the port number from the host byte order to the network byte order.

So, here is an example how you can initiate the address variable. So, you setup the port at 3028 which is you are taking it as an integer variable then you have this sin family AF INET that I have discussed, sin address dot addresses inaddr any to take the local address. If you want you can also put some IP address there but that IP address need to be matched with the IP address used by your network interface. And then in the sin port you make this call to h to ns over the port number to convert it to the network byte order.

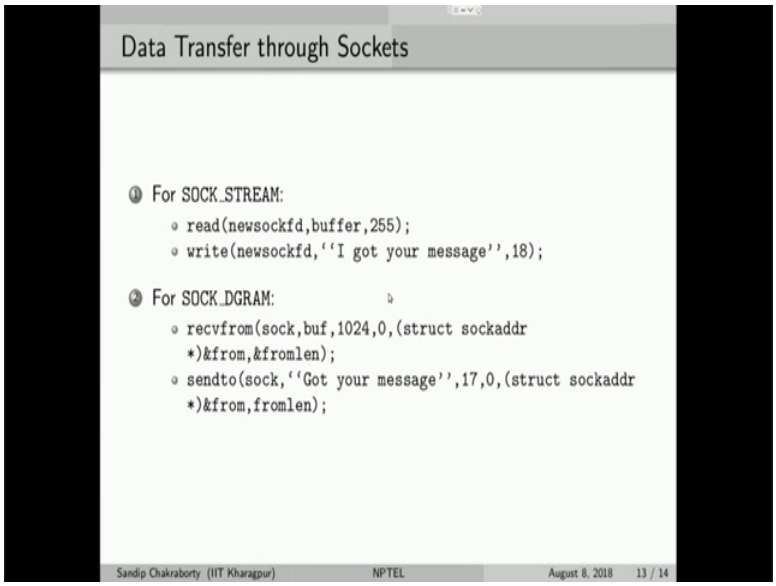
(Refer Slide Time: 22:15)



Well, now to accept a socket connection, so you in the client side you create a client address. Now, the server it is listening on this particular socket. So, this listen function has a parameter called 5 here. So, this particular parameter indicates that how many maximum connection can be backlogged when multiple clients are trying to connect to the server. And then you take this size of this address variable that you have declared and make a accept call. So, this accept call will take the sock fd where the socket is listening the client address that will be provided. So, when the client will get connected the address of the client will get stored in this variable, and the length of that client address.

Now, this accept call whenever you are initiating a connection as I have mentioned that the server need to always in the mode where it is waiting for any incoming connection and the clients need to initiate the connection. So, accordingly we have two kind of connection called active open and the passive open. Now, as I have mentioned that the server needs to announce it address remain in the open state and waits for any incoming connection, so that is the kind of passive open. And the client it only opens a connection where there is a need for data transfer that is the kind of active open. And the connection it is initialized by the client.

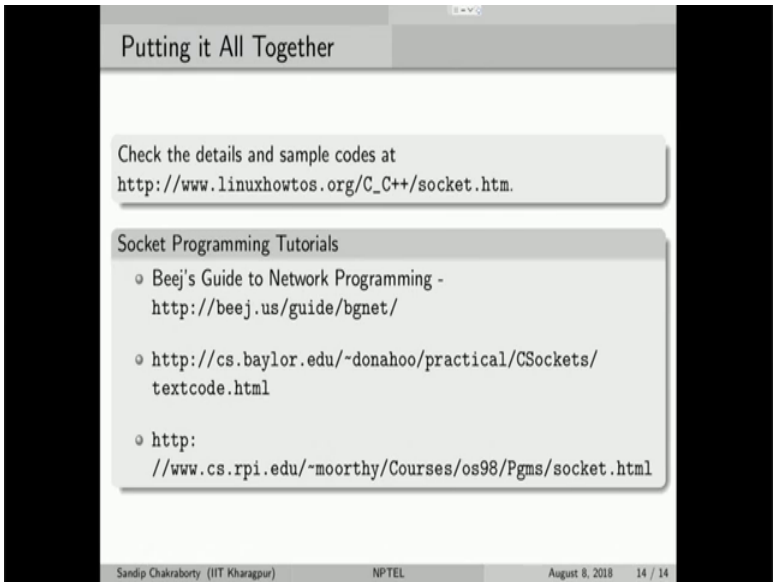
(Refer Slide Time: 23:41)



Now, these are the data transfer format. So, we have two different type of socket; the stream socket and the datagram socket. In case of a stream socket, you can use the function called read and write by providing the socket identifier. So, here is an interesting fact that whenever you are accepting a connection, you are getting a new socket id. So, why you are getting a new socket id, because the server is listening on one socket; now when the client is initiating there can be multiple clients which are connecting simultaneously. Now, when multiple clients are connecting simultaneously, you need to create separate logical pipe to separate client, so that are actually indicated by the socket address which is written by this accept system call.

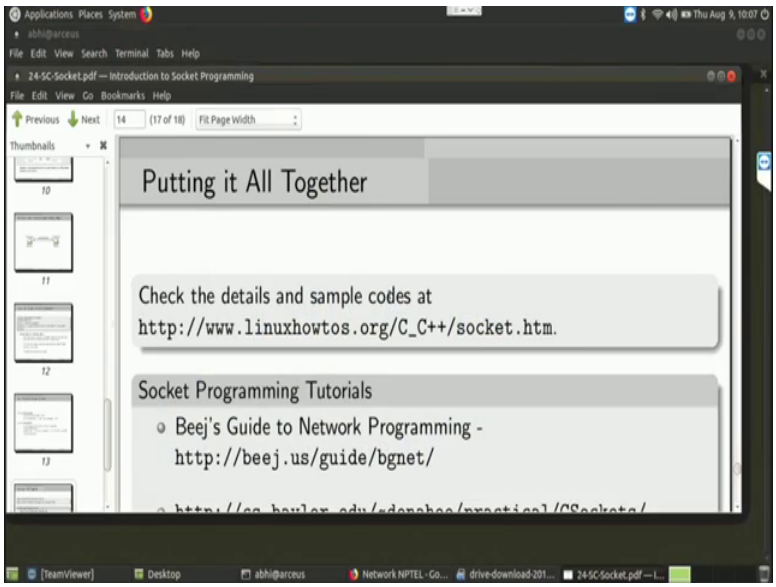
So, the accept system call will return an address and that particular address will be assigned to the new sock fd and while you are sending the data you use that new socket id because that has created end to end stream or end to end pipe to a specific client and you will send a message. So, for the stream kind of socket, you can use the read and the write function to read data from a buffer or to write data to a buffer. For the datagram socket you can use the function called the receive from and send to; receive from function to receive data from the socket or sent to function to send data to that particular socket.

(Refer Slide Time: 25:09)



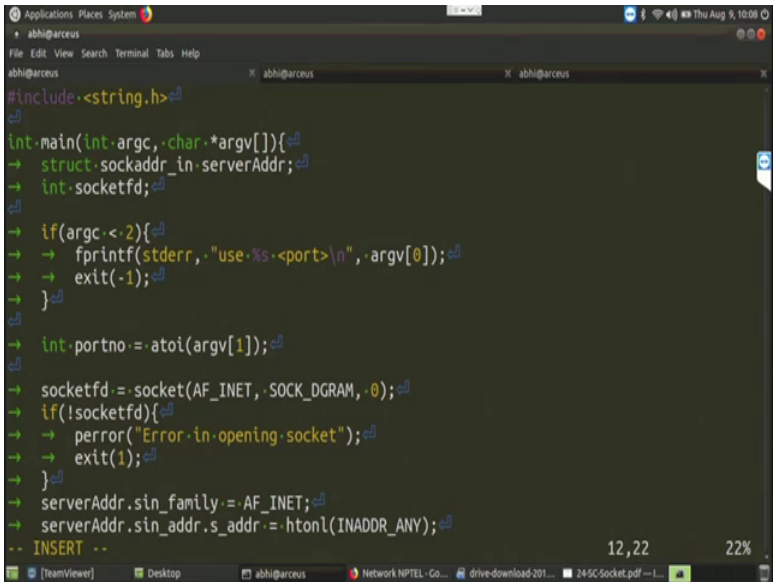
Now, here are some, so I will show you some demo of the entire thing. So, here are some link that you can follow to learn socket programming in more details. So, what I will suggest you to go to these particular links and start writing your own network programming using the socket. So, let us quickly go to some demo of this entire idea of socket programming.

(Refer Slide Time: 25:37)



So, we will first look into UDP server and a corresponding UDP client. So, let us open look into the UDP server first.

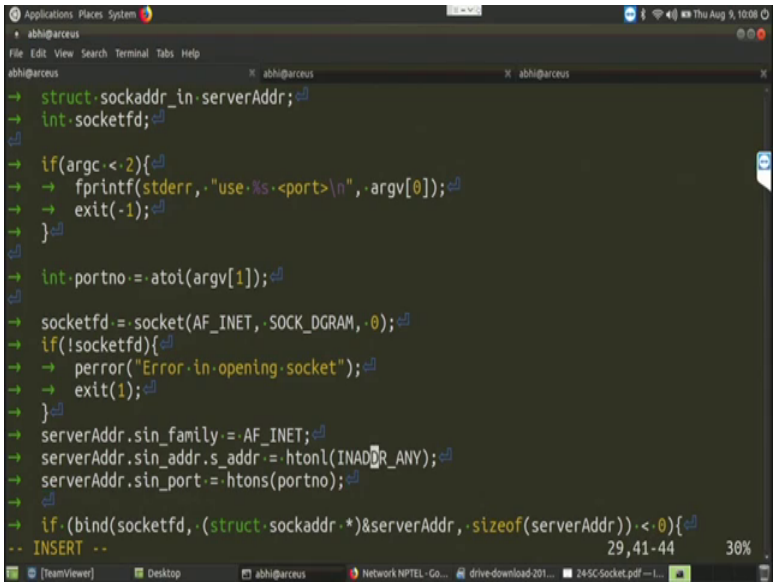
(Refer Slide Time: 26:05)



So, here is your code for the UDP server. So, in the UDP server code, you can see that we have included some header, these are the kind of standard headers that we have to include. And then inside the main function, we are declaring the entire thing. So, in the main function, we have this we are first defining a socket, which is the struct sockaddr in and the corresponding server address. Then we are defining a socket identifier. And we are defining a port number.

Now, here this we are first making a socket system call. In the socket system, call you have this AF INET the parameter that we have mentioned we need to specify the data gram socket. We are specifying because we are trying to create a UDP socket and final parameter is equal to 0. The 0 parameter that we send for the protocol field that we have mentioned. Then once the socket is created, if there is an error, we print some error message.

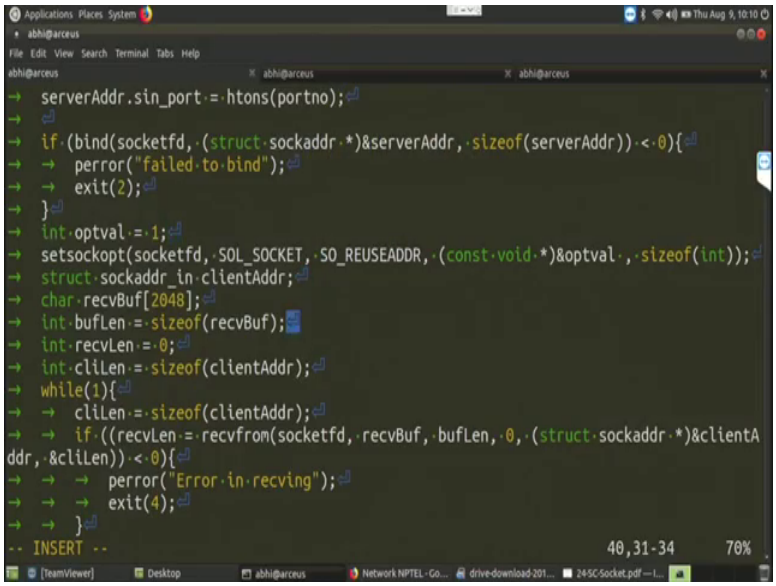
(Refer Slide Time: 27:17)



Otherwise we declare the server address as we have discussed earlier. So, we declare it as the protocol family as AF INET the address as inaddr any with a address of this machine and then the port number. Then we make the bind system call. The bind system call is to bind the socket with the corresponding port number that we are specifying as a command line argument. And finally, we make a call to a function called setsockopt, this setsockopt is to set some option to the socket, here we have set the option is so underscore reuse Addr this. So, underscore reuse Addr will help us to use the same port for multiple connections together and that is not a safe idea, but sometime you can use that.

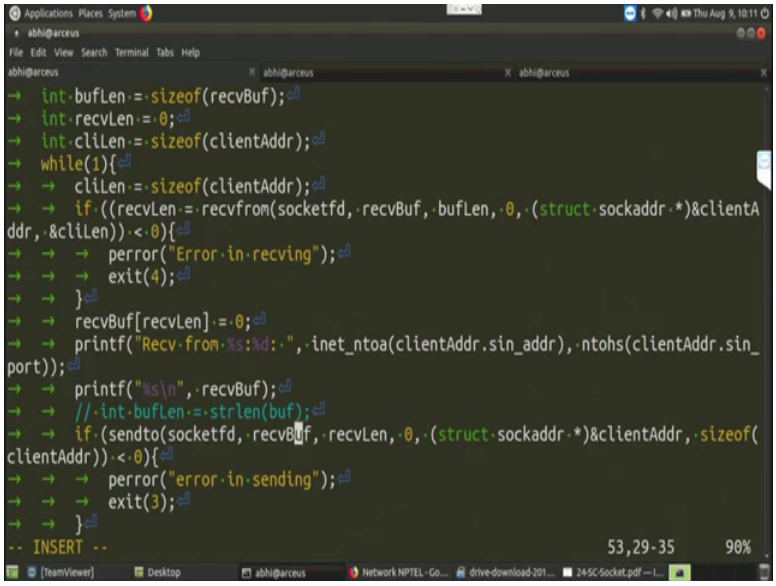
Then after that we are declaring the buffer where we will store the data. We are declaring the receive buffer and the length of the data. We are again declaring a address for the client that we have shown; the address where the client variable will get stored. And after that we are making this receive from function. Now, you see for the UDP case, we do not create any connection. So, we do not require this connect and accept calls. So, the connect and accept call are specific to the TCP server; for the UDP server you do not require the connect and accept call, because we do not have a connection establishment in case of UDP.

(Refer Slide Time: 29:10)

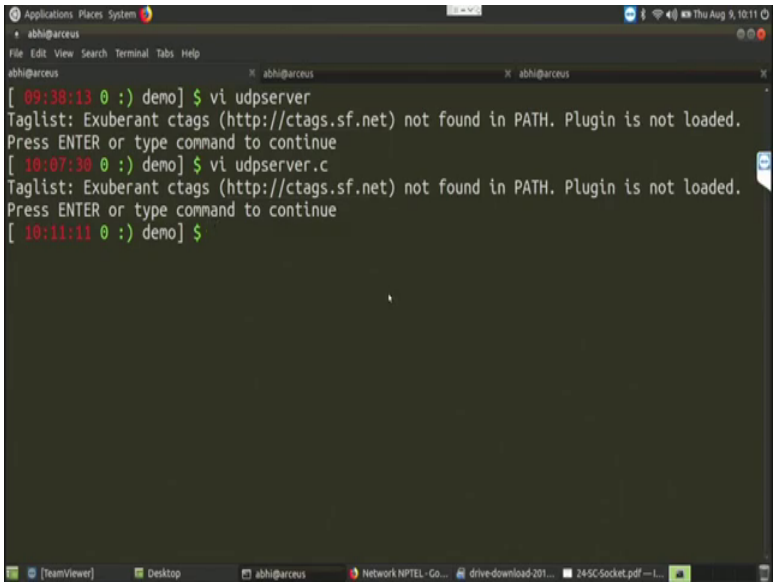


So, what you can do once you have created the socket at the server side, you can directly make the call to the receive from function to receive the data. And this receive from function will contain the client details. So, this is the client address which is clear in the receive form function. And once you are receiving data we are printing some data and making a send to call. So, this send to call is sending the data to the corresponding client, so that is the port at the server side.

(Refer Slide Time: 29:28)

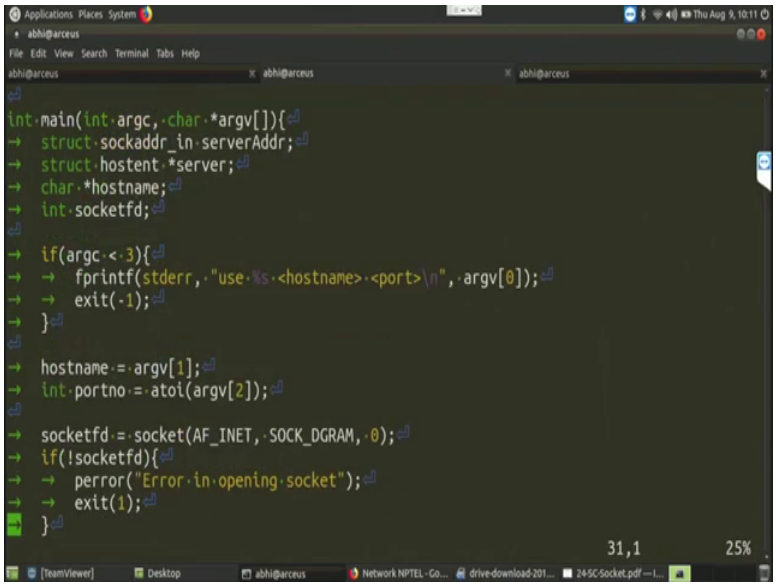


(Refer Slide Time: 29:37)



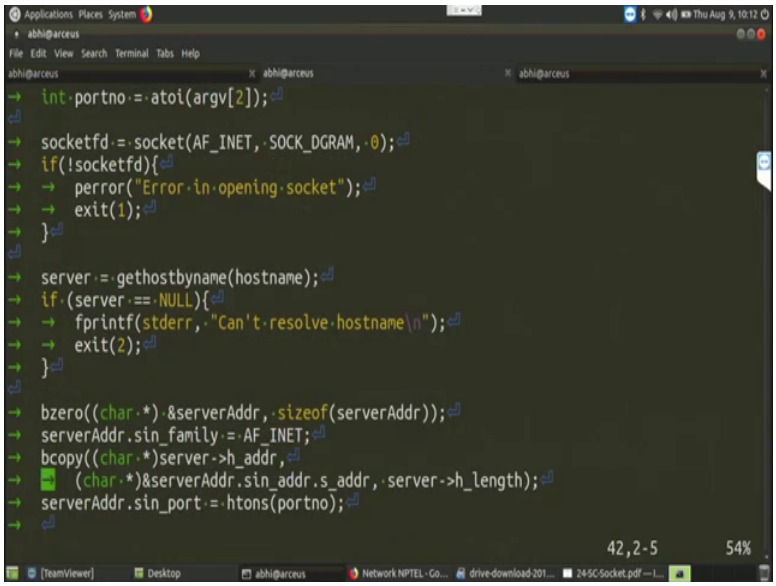
Now, let us open the client side code for the UDP. So, this is the client side code for the UDP. At the client side, we do in the same way we declared the address the server address where we want to connect.

(Refer Slide Time: 30:00)



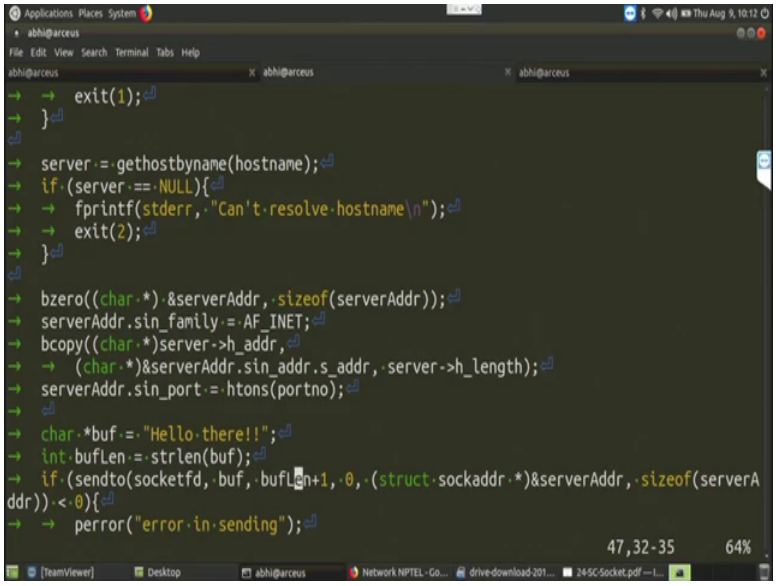
So, then from the comment line, we take the hostname and a port, the host name of the name of the server and the port address where the server is getting binded. Then we create a socket with AF INET and as a datagram socket at the client side as well. Then we get the server IP.

(Refer Slide Time: 30:25)



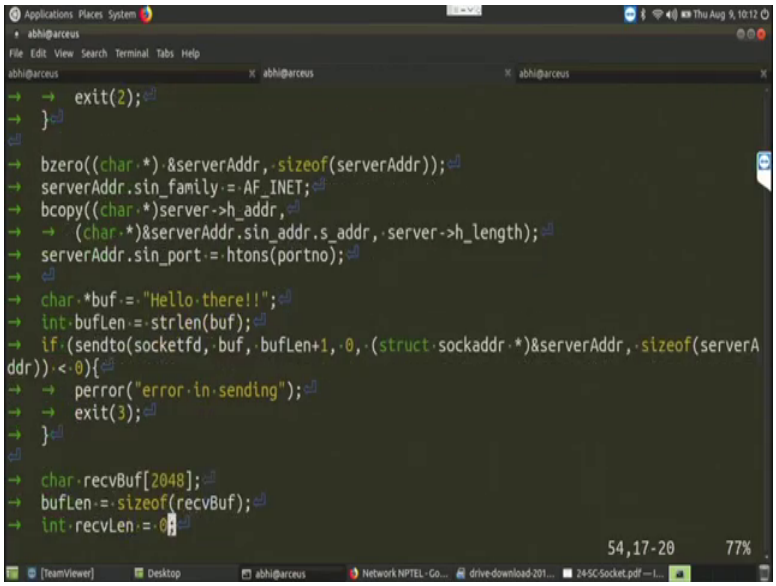
After that we are creating the server address by using that setting the sin family the host address and the server port. So, the value which are being provided by the by the client at the through the comment line. So, after that once you have got the server address, again you do not need to initiate a connection, you can directly make the send to call to send some data. So, from here we are making a send to call. So, this send to call is sending the data to the particular socket with this server address which we are specifying here.

(Refer Slide Time: 30:53)



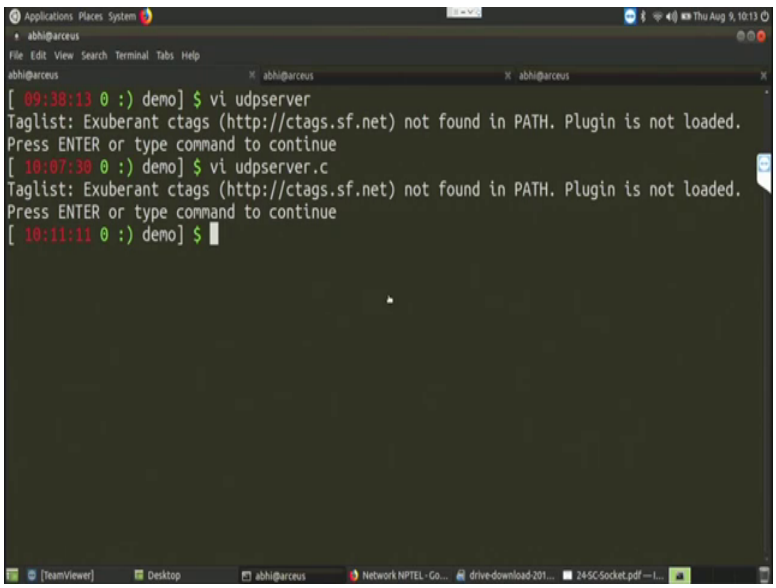
And after that you are sending the data, we are receiving certain data.

(Refer Slide Time: 31:07)



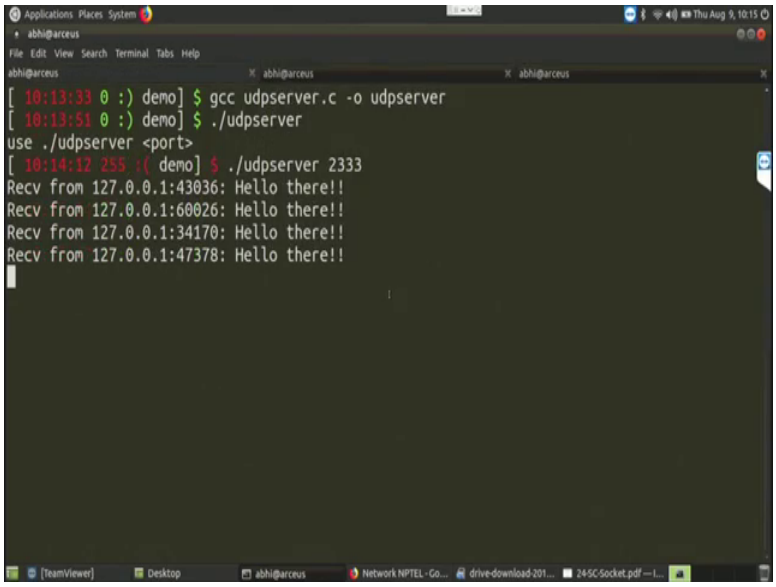
So, we are sending the data from the client as this message hello dear and from the client side we are receiving the data. So, we are receiving the data in the form of what is returned back by the sender, we have declared a temporary character buffer here a stream here stream buffer here. And we have made this receive from call to receive the data from the client side from a server side. So, here the server address is being provided, so that is and then we are printing the data here, so that is the client side code. Now, let us run the server and the client side code. Here in the same machine we are running the server side code and a client side code.

(Refer Slide Time: 31:51)



So, first let us compile the server code and the client code.

(Refer Slide Time: 32:00)



So, now let us first run the server. So, according to our syntax we that is we have to run the server and specify a port address where the server will connect itself the bind through the bind system call. So, let us give the port number as 2333. So, the server is now running. Now, from the client side we can run the client. And we are running the server in the same machine, so the host name of the server you can give it as local host. And to you are running the server at the port 2333; at port 2333, so we can provide the server port as 2333. So, it has send a message to the server you can see that the server has received the message from the client and it is returning back that message. So, the client has received the message and printed it.

So, you can again run the client. And you can see that it has received a message. Now, note the note one thing here you are printing the server IP the client IP and the client port, the client IP is the local IP of this machine. And a client port whenever we are running multiple client, the client code gets changed. Now, at the client side as the client do not bind itself to our well known port during the runtime, the client randomly chooses one port at this and initiate the client transfer from that particular port address, so that is why a different run the port address gets changed.

So, if I run it again multiple time at different time, the port address gets changed. So, at different instances it takes different port. So, this is a demo about the UDP server and a

UDP client which is the possibly the simplest form of the socket programming. We will share the code with you, we will request you to browse to the code run into your own machine and see what is happening and understand it more details. So, in the next class, we will show you the demo about the TCP server, TCP client and some variants of TCP server and the TCP client.

Thank you all for attending the class.