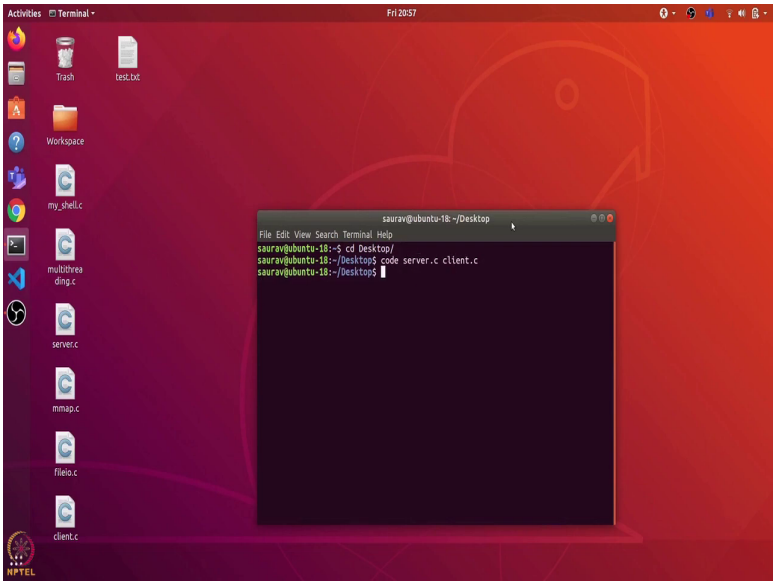
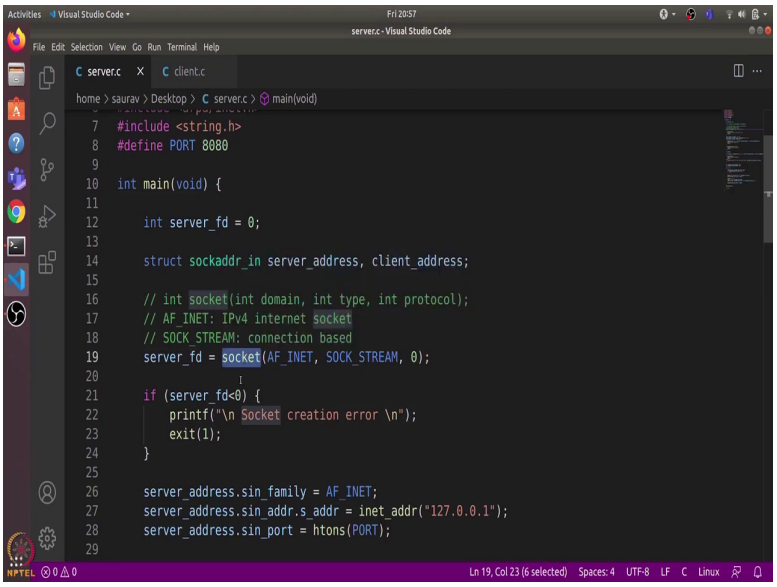
Design and Engineering of Computer Systems
Professor Mythili Vutukuru
Computer Science and Engineering
Indian Institute of Technology, Bombay
Lecture 30 (Week - 4, Tutorial - 2)
Socket programming

(Refer Slide Time: 0:15)

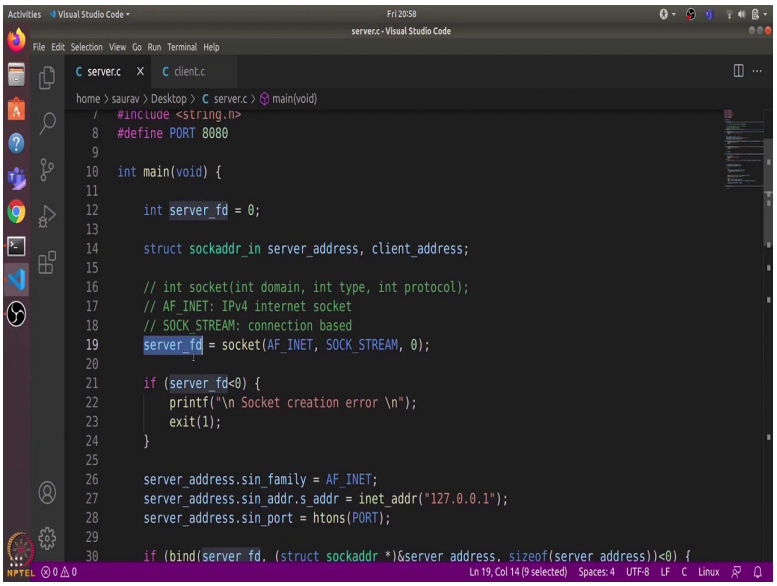




Hi everyone. In this video, we will learn about socket programming. So, I have written this server.c and client.c programs. Let us open them in visual code. So, let us have a look at the server code first. So, here is the main program. First of all, we use this socket system call to create a new socket.

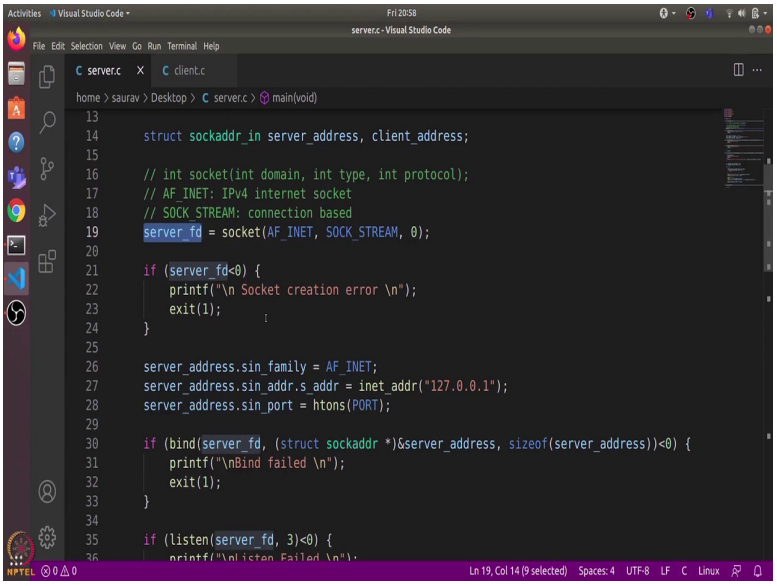
And its first argument is the domain, which specifies that this is an internet socket. The second argument specifies what is the type of socket, so we have mentioned sock stream, which means that it will be a connection based socket. And the last argument specifies the protocol, we have used 0 here, so it will automatically decide the protocol type based on the sock stream. So, this returns a socket file descriptor.

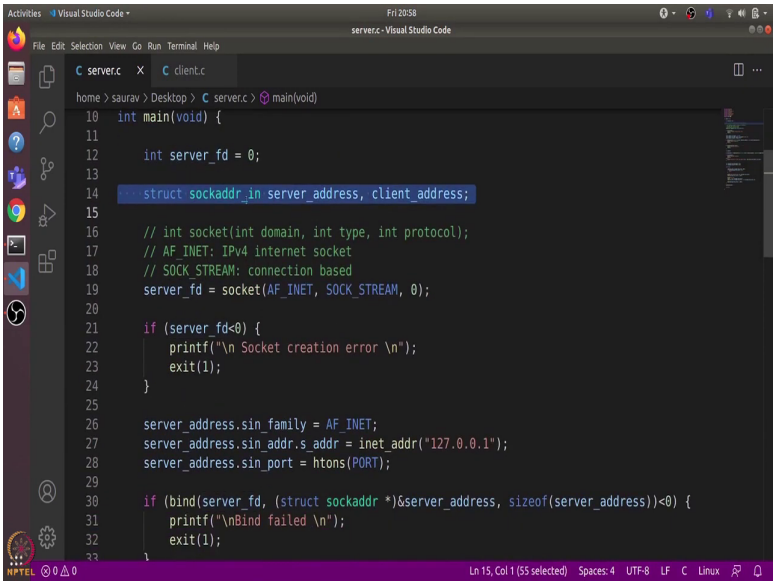
(Refer Slide Time: 1:15)

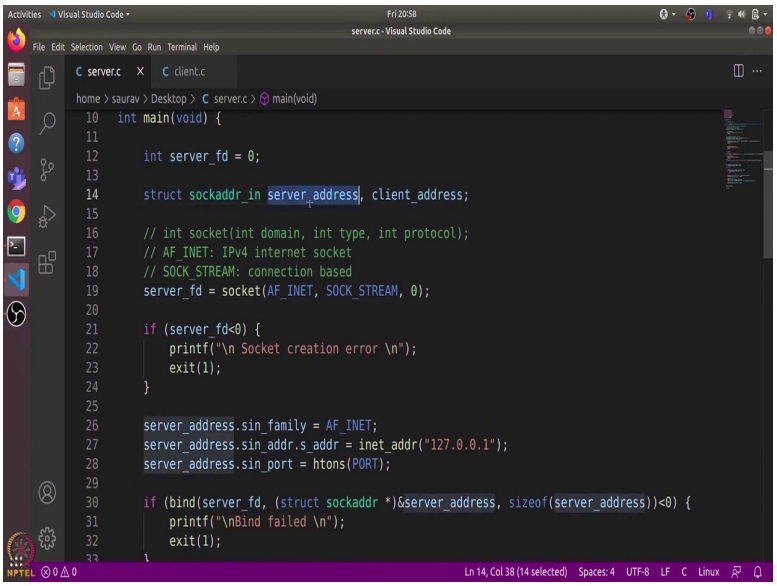


And we store it in the server file descriptor variable, then we check if it has returned a negative value. In that case, there is an error and we exit the program.

(Refer Slide Time: 1:25)

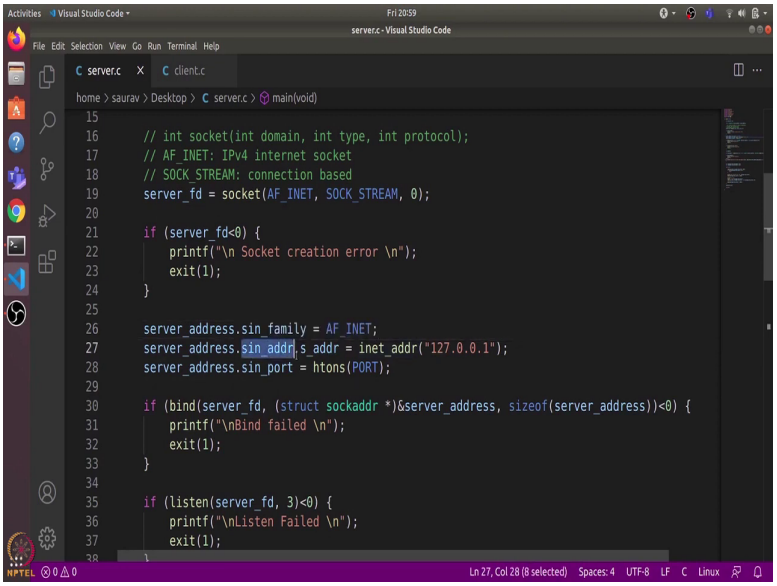


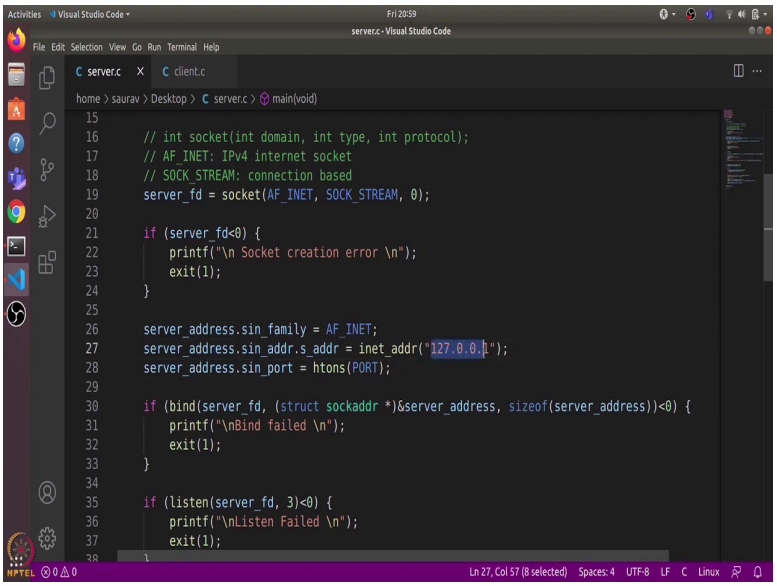


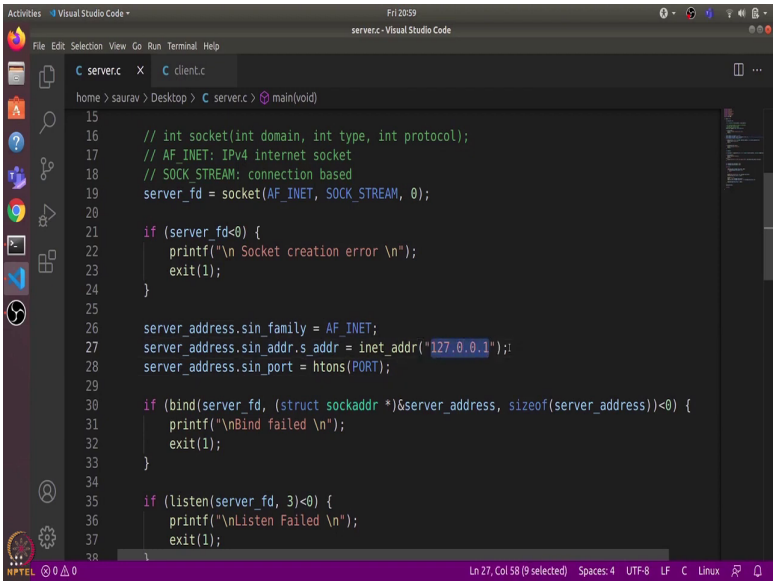


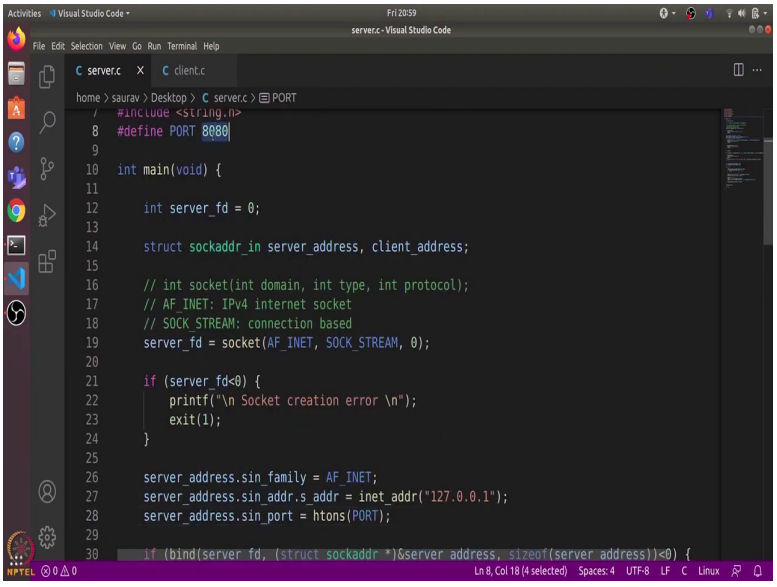
We now bind this socket file descriptor with an address. So, in server program, first we create a socket, and then we bind it with an address so that clients can connect to this socket using that address. So, we declare a socket address struct, and we call it server address.

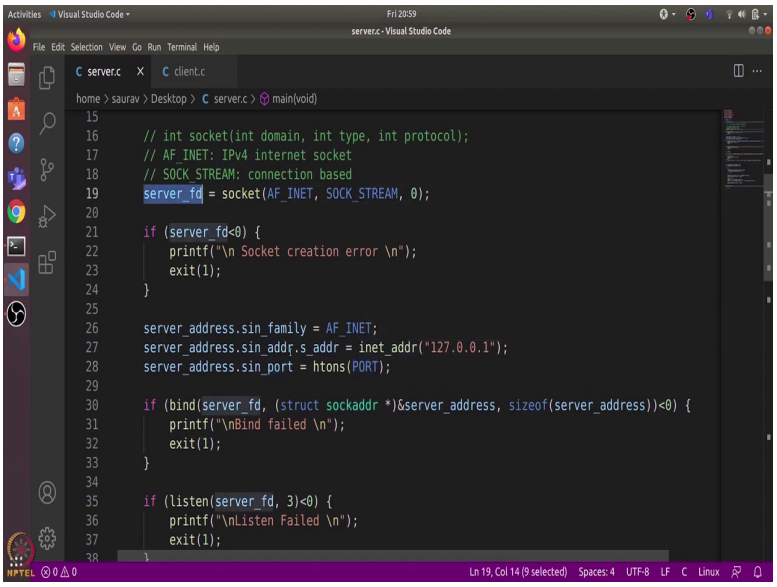
(Refer Slide Time: 1:44)





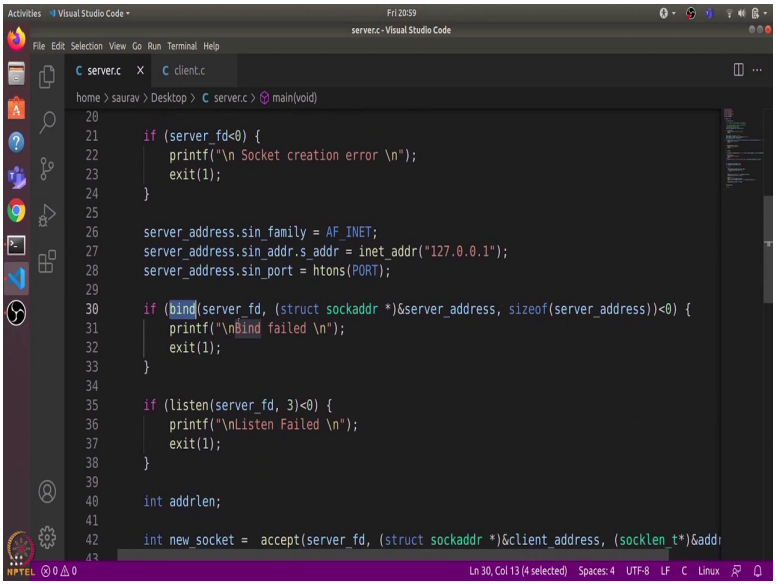

Activities
Visual Studio Code
Fri 20:59
server.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > server.c > main(void)
// int socket(int domain, int type, int protocol);
// AF_INET: IPv4 internet socket
// SOCK_STREAM: connection based
server_fd = socket(AF_INET, SOCK_STREAM, 0);
if (server_fd<0) {
printf("\n Socket creation error \n");
exit(1);
}
server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
server_address.sin_port = htons(PORT);
if (bind(server_fd, (struct sockaddr *)&server_address, sizeof(server_address))<0) {
printf("\nBind failed \n");
exit(1);
}
if (listen(server_fd, 3)<0) {
printf("\nListen Failed \n");
exit(1);
}
Ln 27, Col 58 (9 selected)
Spaces: 4
UTF-8
LF
C
Linux

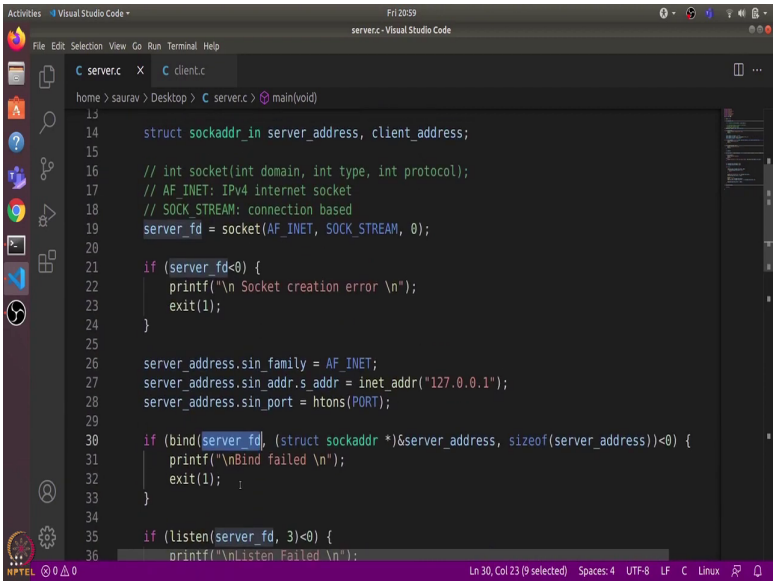

Activities
Visual Studio Code
Fri 20:59
server.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > server.c > PORT
#include <string.h>
#define PORT 8080
int main(void) {
int server_fd = 0;
struct sockaddr_in server_address, client_address;
// int socket(int domain, int type, int protocol);
// AF_INET: IPv4 internet socket
// SOCK_STREAM: connection based
server_fd = socket(AF_INET, SOCK_STREAM, 0);
if (server_fd<0) {
printf("\n Socket creation error \n");
exit(1);
}
server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
server_address.sin_port = htons(PORT);
if (bind(server_fd, (struct sockaddr *)&server_address, sizeof(server_address))<0) {
Ln 8, Col 18 (4 selected)
Spaces: 4
UTF-8
LF
C
Linux

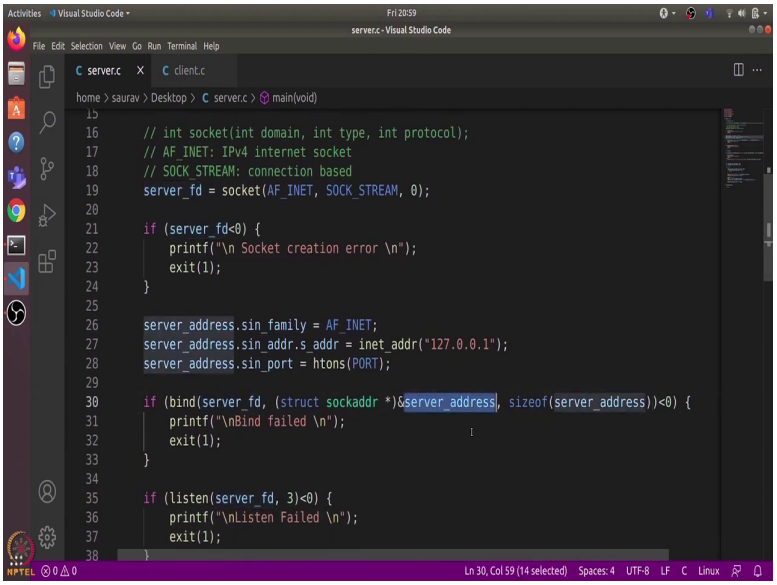

Activities
Visual Studio Code
Fri 20:59
server.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > server.c > main(void)
// int socket(int domain, int type, int protocol);
// AF_INET: IPv4 internet socket
// SOCK_STREAM: connection based
server_fd = socket(AF_INET, SOCK_STREAM, 0);
if (server_fd<0) {
printf("\n Socket creation error \n");
exit(1);
}
server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
server_address.sin_port = htons(PORT);
if (bind(server_fd, (struct sockaddr *)&server_address, sizeof(server_address))<0) {
printf("\nBind failed \n");
exit(1);
}
if (listen(server_fd, 3)<0) {
printf("\nListen Failed \n");
exit(1);
}
Ln 19, Col 14 (9 selected)
Spaces: 4
UTF-8
LF
C
Linux

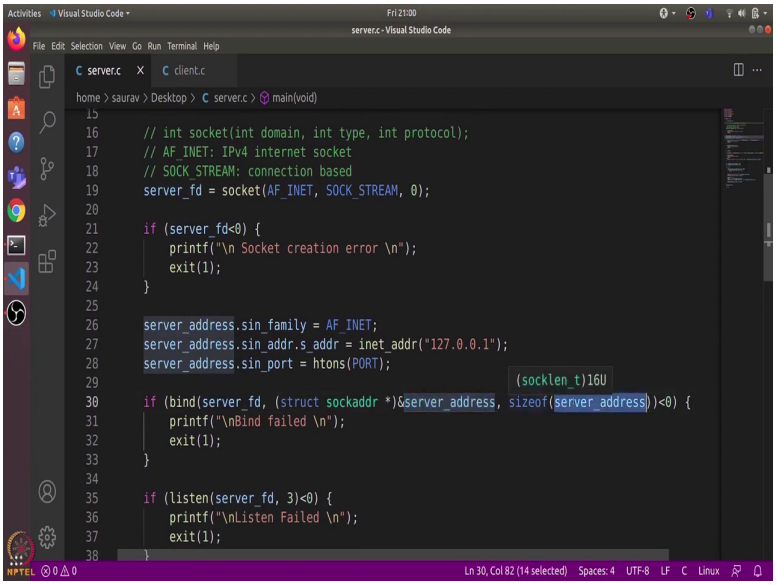
We define the family as AF_INET, as we are using the internet sockets, then we define the address as 127.0.0.1. So, this is the address of the localhost, because we will run both client and server on the same machine. So, we will use the localhost IP address. And then we also need to specify the port number. Here we are using 8080 as the port. So, now that we have specified both the IP address and the port, we will bind the socket that we have created with this address.

(Refer Slide Time: 2:19)



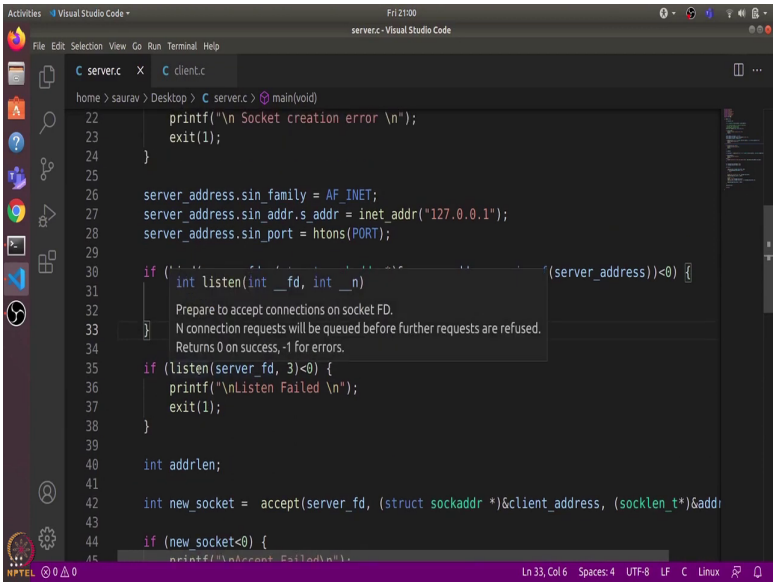


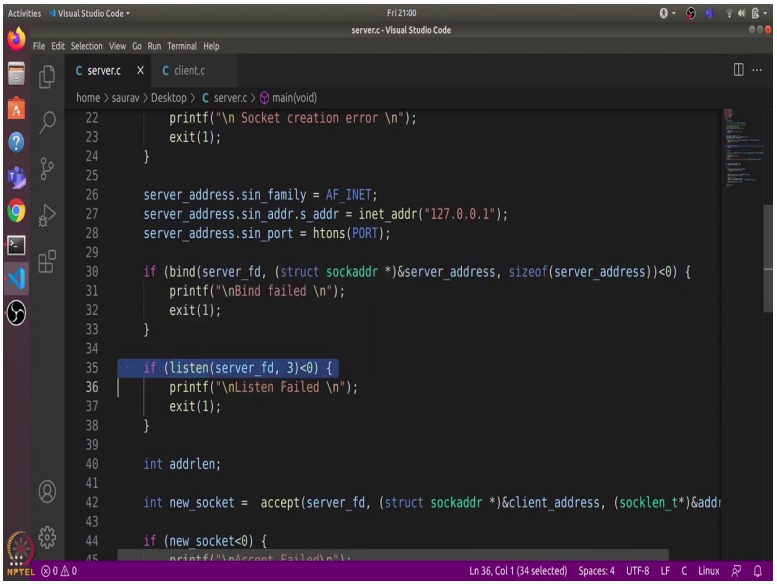


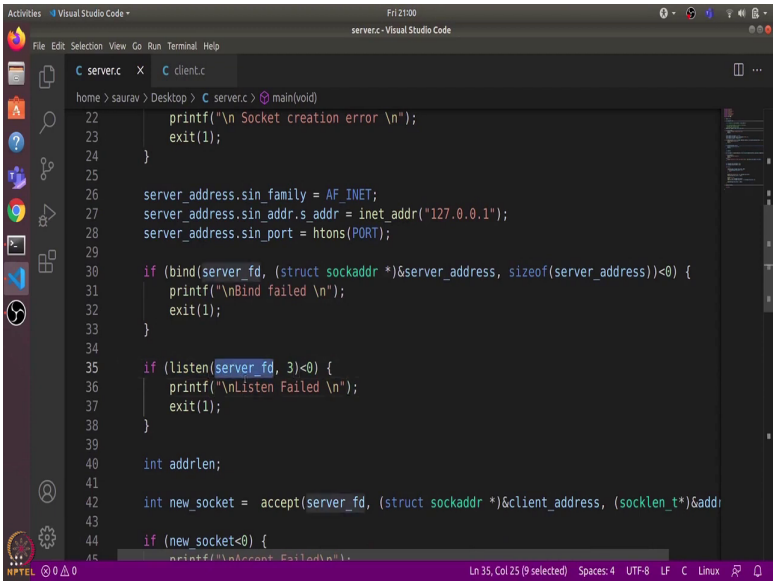


So, we will use the bind system call. It takes the socket file descriptor. And it also takes the address struct as the second argument and the size of address struct as the third argument. And if it returns negative value that means there is some error, otherwise, the socket is successfully bound to the address mentioned here.

(Refer Slide Time: 2:38)

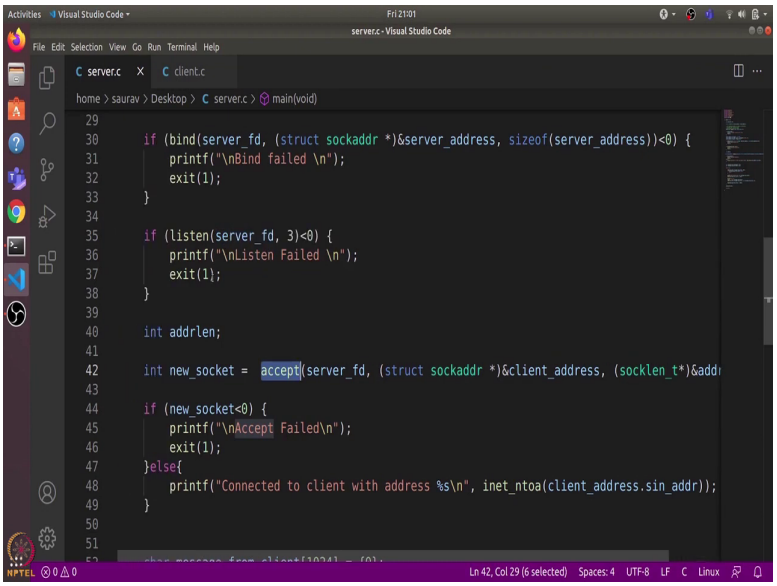


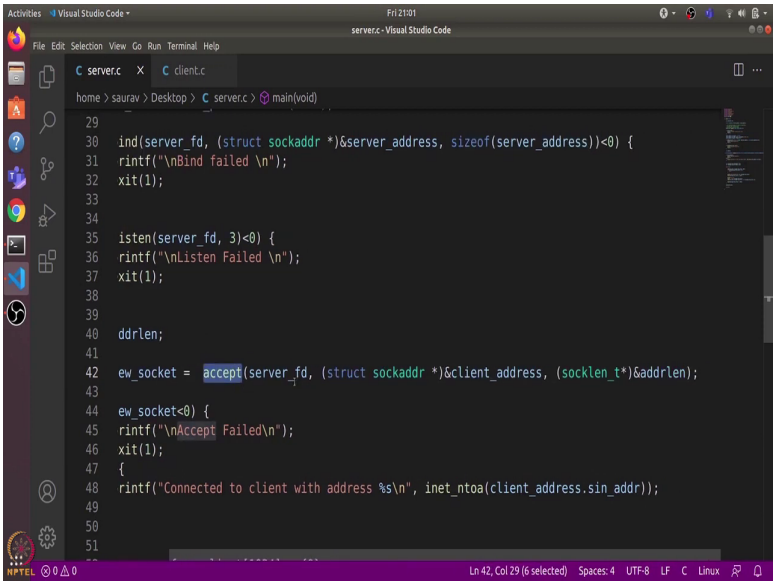




So, after we have assigned the address to this socket, now we can use the Listen system call to listen for incoming connection requests to this socket. This listen system call takes two arguments first is the socket file descriptor and second is the number of connections which will be waiting in the queue.

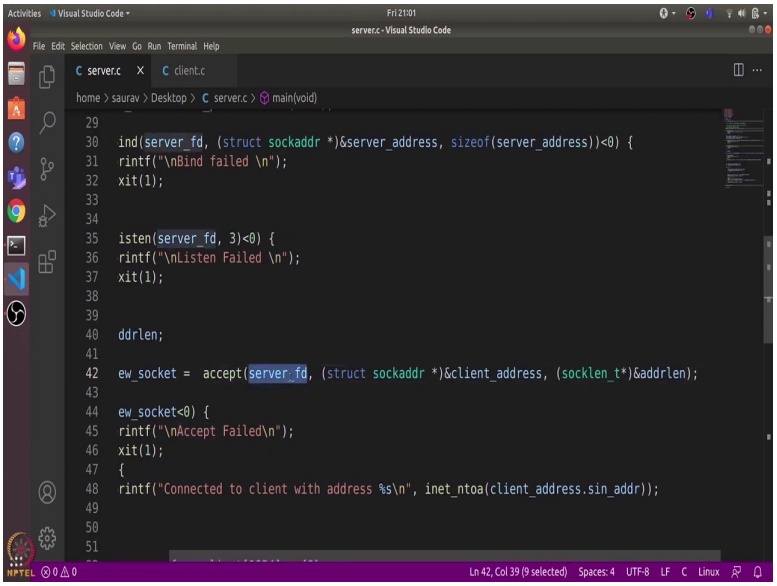
(Refer Slide Time: 2:54)

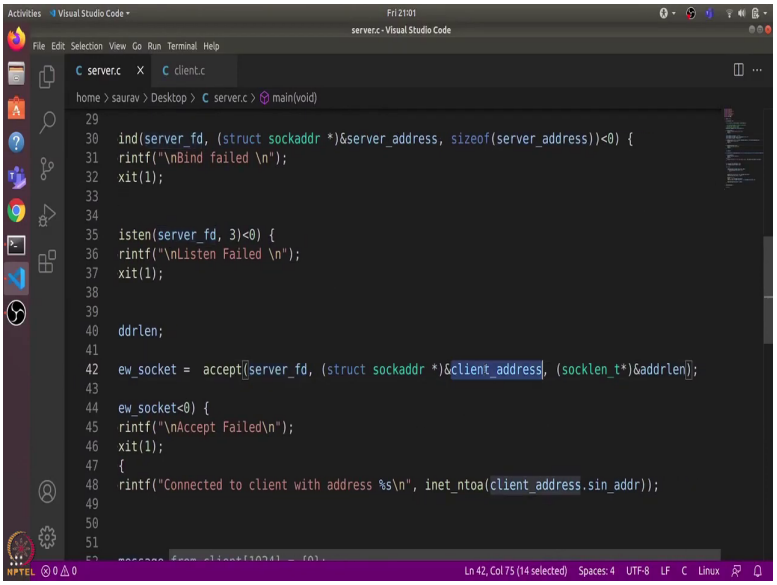


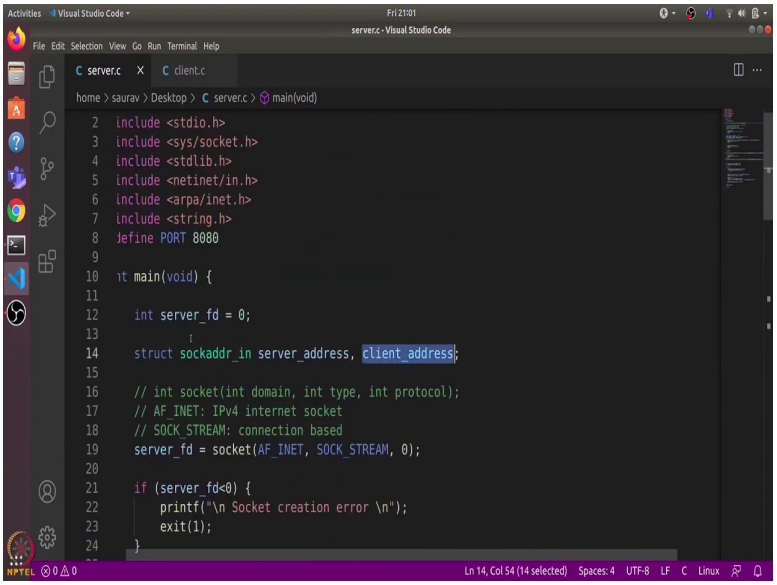


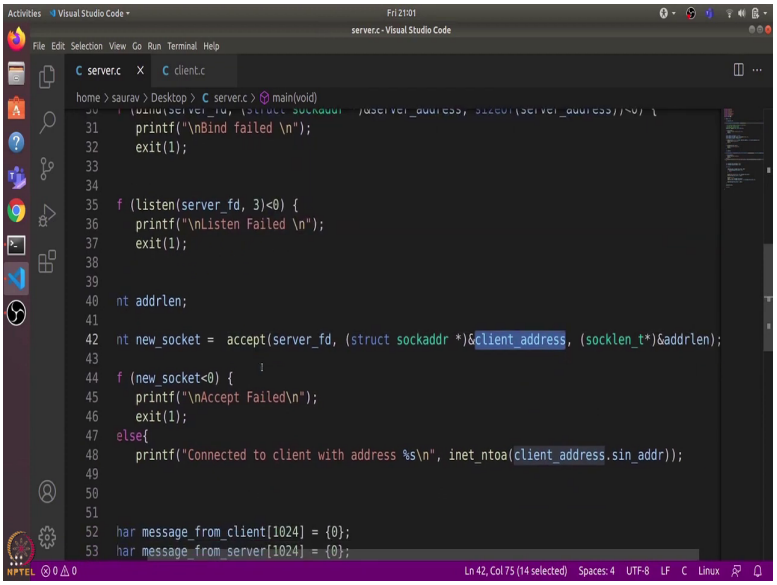
So, this system call also if it returns a negative value that means there is some error. Otherwise, now we accept the connections on this socket. So, we use this accept system call to accept incoming connections. This accept system call will take in the first client request which is waiting in the queue and connect the socket to that client. So, this accept system call takes in three arguments.

(Refer Slide Time: 3:24)



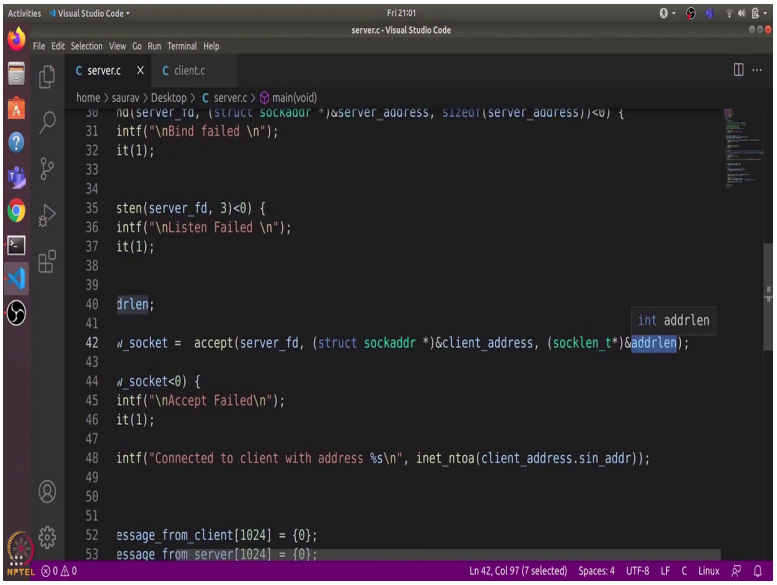


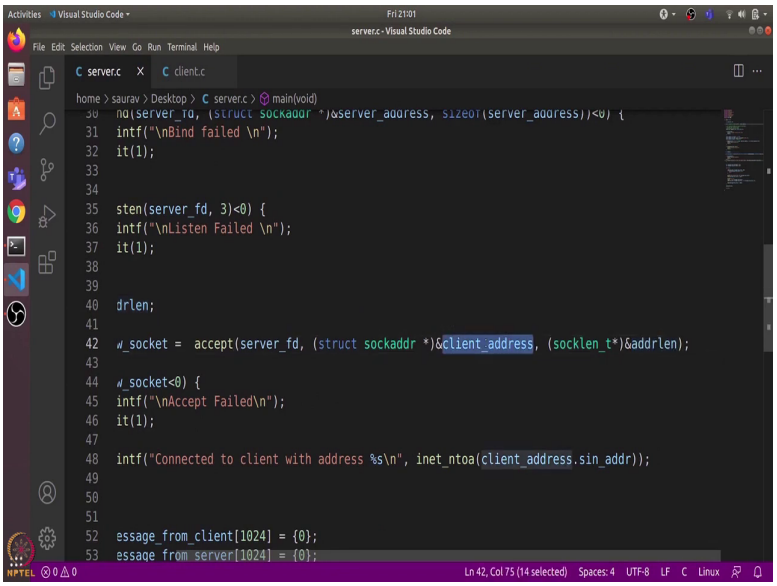


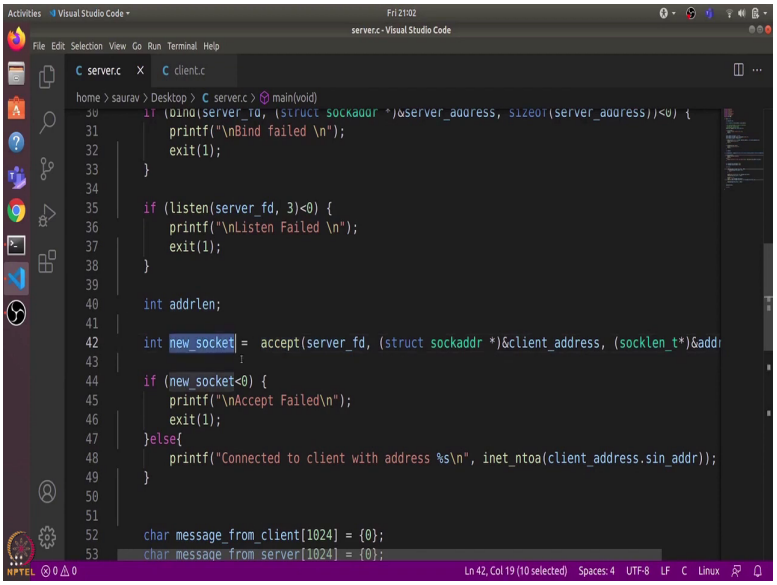


First is the socket file descriptor. And second is a client address struct, which initially is empty, we have just declared a structure. And we give it an address as an argument to accept system call. And this will fill in the client details in this struct.

(Refer Slide Time: 3:41)

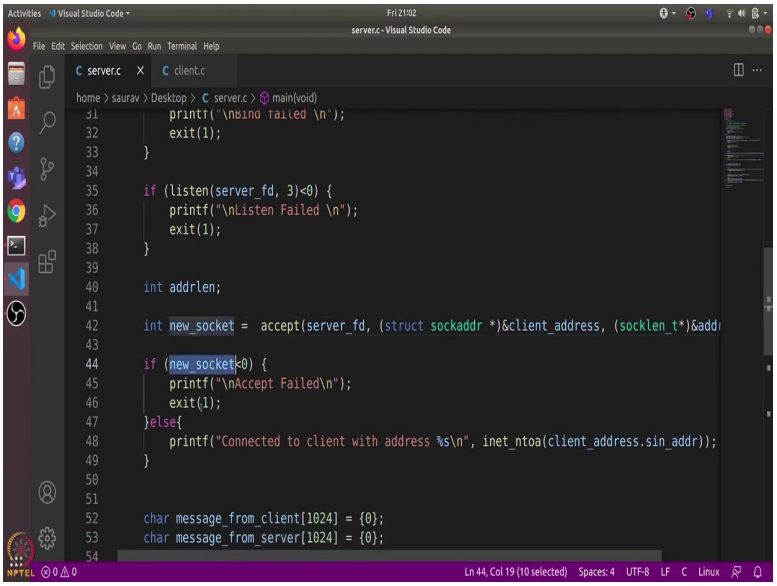

Activities
Visual Studio Code
Fri 21:01
server.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > server.c > main(void)
intf("\nBind failed \n");
it(1);
sten(server_fd, 3)<0) {
intf("\nListen Failed \n");
it(1);
drlen;
int addrlen
w_socket = accept(server_fd, (struct sockaddr *)&client_address, (socklen_t*)&addrlen);
w_socket<0) {
intf("\nAccept Failed\n");
it(1);
intf("Connected to client with address %s\n", inet_ntoa(client_address.sin_addr));
essage_from_client[1024] = {0};
essage from server[1024] = {0};
Ln 42, Col 97 (7 selected)
Spaces: 4
UTF-8
LF
C
Linux

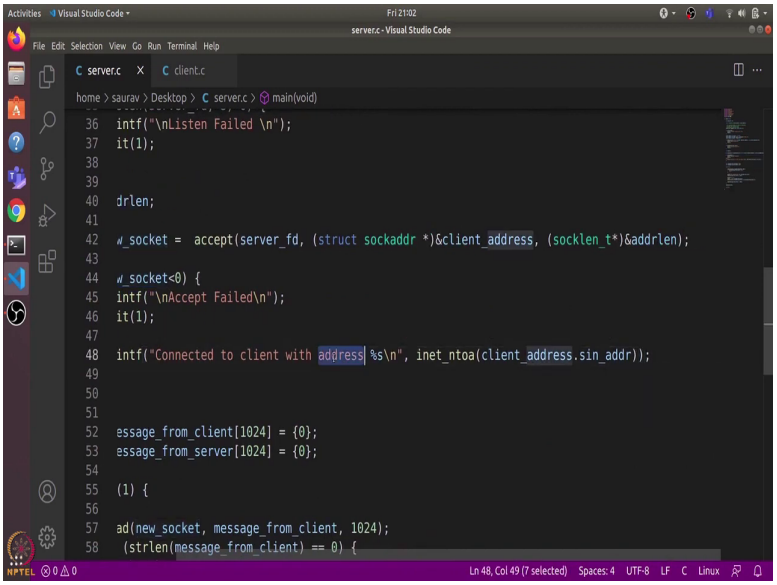

Activities
Visual Studio Code
Fri 21:01
server.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > server.c > main(void)
intf("\nBind failed \n");
it(1);
sten(server_fd, 3)<0) {
intf("\nListen Failed \n");
it(1);
drlen;
w_socket = accept(server_fd, (struct sockaddr *)&client_address, (socklen_t*)&addrlen);
w_socket<0) {
intf("\nAccept Failed\n");
it(1);
intf("Connected to client with address %s\n", inet_ntoa(client_address.sin_addr));
essage_from_client[1024] = {0};
essage from server[1024] = {0};
Ln 42, Col 75 (14 selected)
Spaces: 4
UTF-8
LF
C
Linux

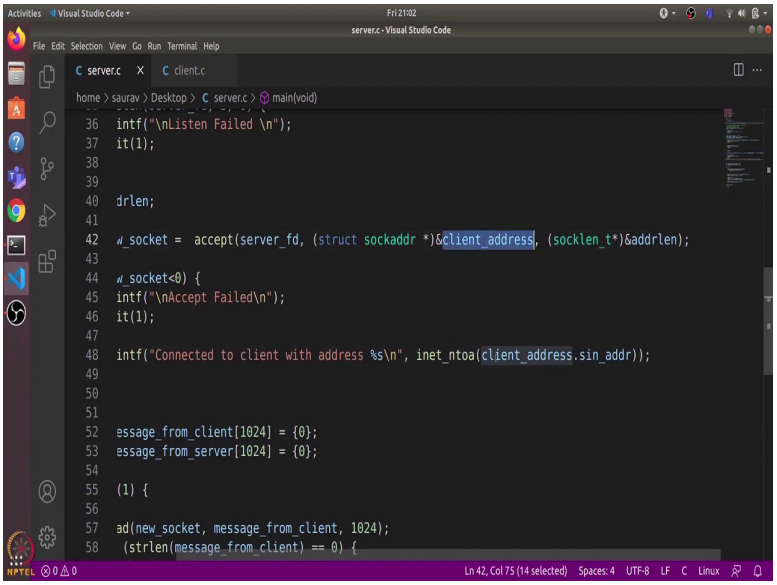

Activities
Visual Studio Code
Fri 21:02
server.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > server.c > main(void)
printf("\nBind failed \n");
exit(1);
if (listen(server_fd, 3)<0) {
printf("\nListen Failed \n");
exit(1);
int addrlen;
int new_socket = accept(server_fd, (struct sockaddr *)&client_address, (socklen_t*)&addr
if (new_socket<0) {
printf("\nAccept Failed\n");
exit(1);
}else{
printf("Connected to client with address %s\n", inet_ntoa(client_address.sin_addr));
char message_from_client[1024] = {0};
char message_from_server[1024] = {0};
Ln 42, Col 19 (10 selected)
Spaces: 4
UTF-8
LF
C
Linux

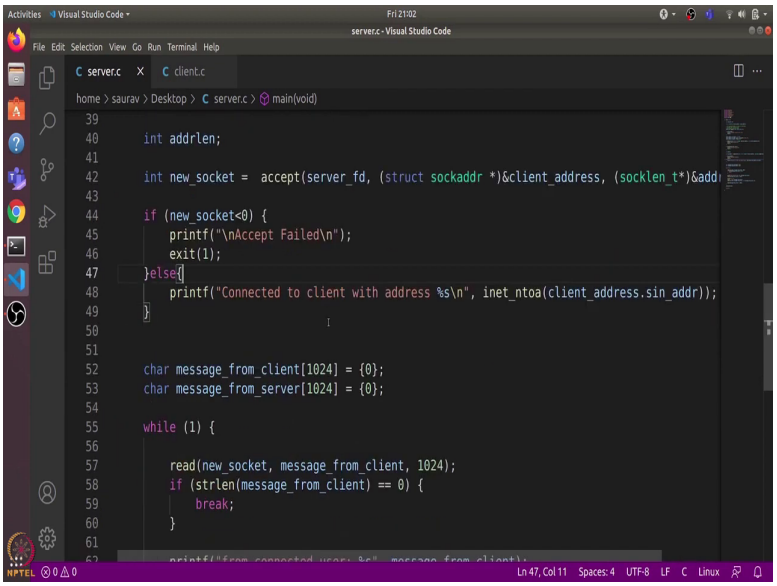
Also, it takes in a address to this int variable, and it will store the size of this struct in the address length variable. And this accept system call returns a new socket file descriptor which is specific to this client.

(Refer Slide Time: 3:58)



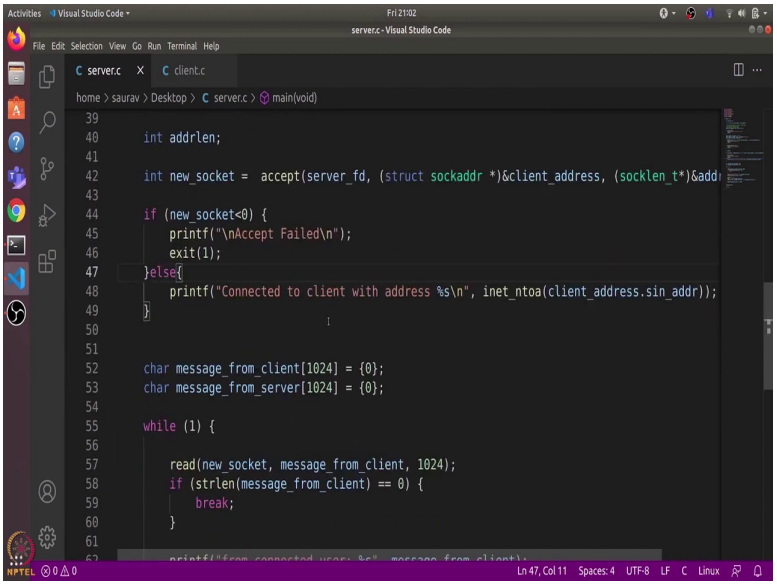


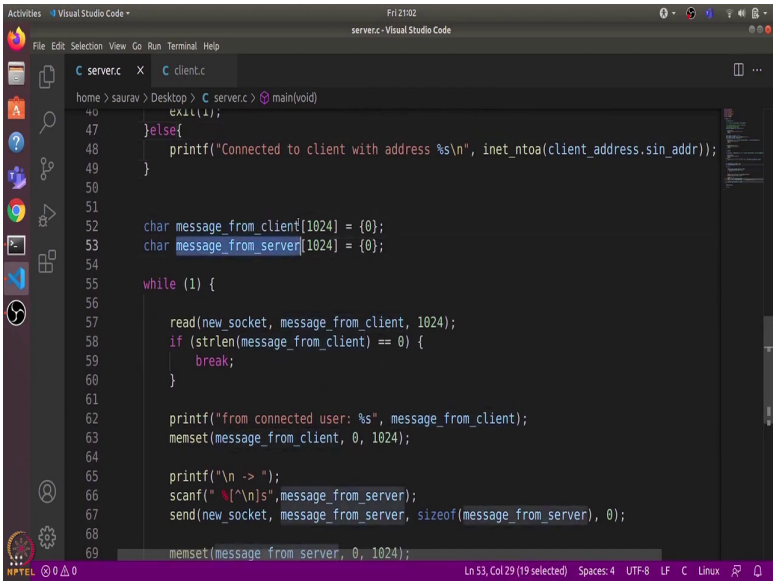


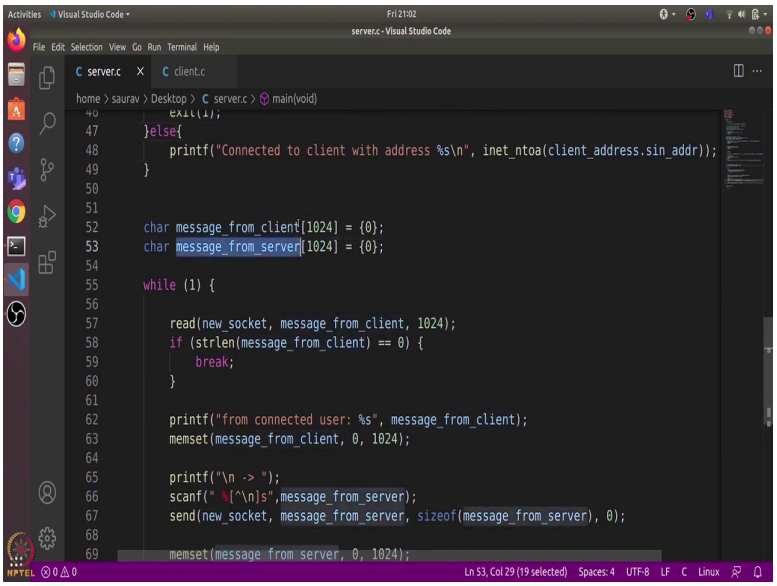


So, if this new socket is less than 0, which means there is some error, otherwise, we print out that we are connected to the client. And we take the client address from this struct. And we use this inet_ntoa to convert this to the ASCII format.

(Refer Slide Time: 4:12)

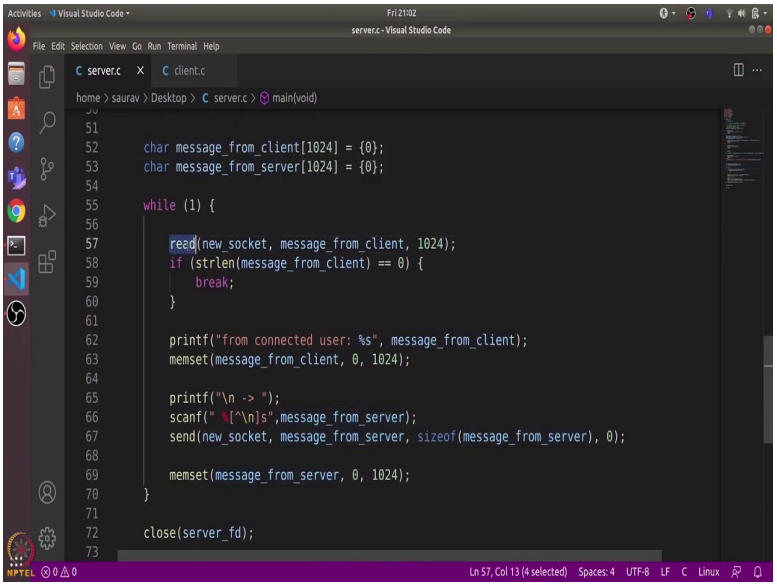

Activities
Visual Studio Code
Fri 21:02
server.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > server.c > main(void)
int addrlen;
int new_socket = accept(server_fd, (struct sockaddr *)&client_address, (socklen_t*)&addr
if (new_socket<0) {
printf("\nAccept Failed\n");
exit(1);
}else{
printf("Connected to client with address %s\n", inet_ntoa(client_address.sin_addr));
}
char message_from_client[1024] = {0};
char message_from_server[1024] = {0};
while (1) {
read(new_socket, message_from_client, 1024);
if (strlen(message_from_client) == 0) {
break;
}
Ln 47, Col 11
Spaces: 4
UTF-8
LF
C
Linux

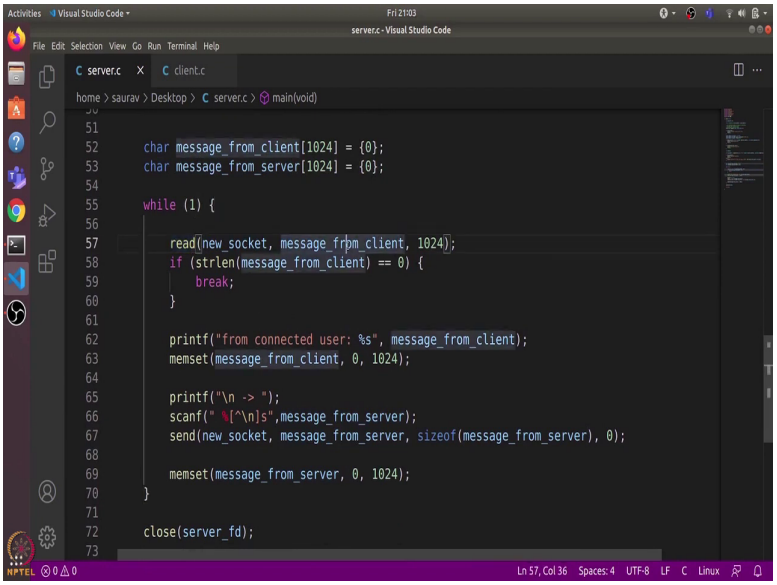

Activities
Visual Studio Code
Fri 21:02
server.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > server.c > main(void)
}else{
printf("Connected to client with address %s\n", inet_ntoa(client_address.sin_addr));
}
char message_from_client[1024] = {0};
char message_from_server[1024] = {0};
while (1) {
read(new_socket, message_from_client, 1024);
if (strlen(message_from_client) == 0) {
break;
}
printf("from connected user: %s", message_from_client);
memset(message_from_client, 0, 1024);
printf("\n -> ");
scanf(" %[^\n]s",message_from_server);
send(new_socket, message_from_server, sizeof(message_from_server), 0);
memset(message_from_server, 0, 1024);
Ln 53, Col 29 (19 selected)
Spaces: 4
UTF-8
LF
C
Linux

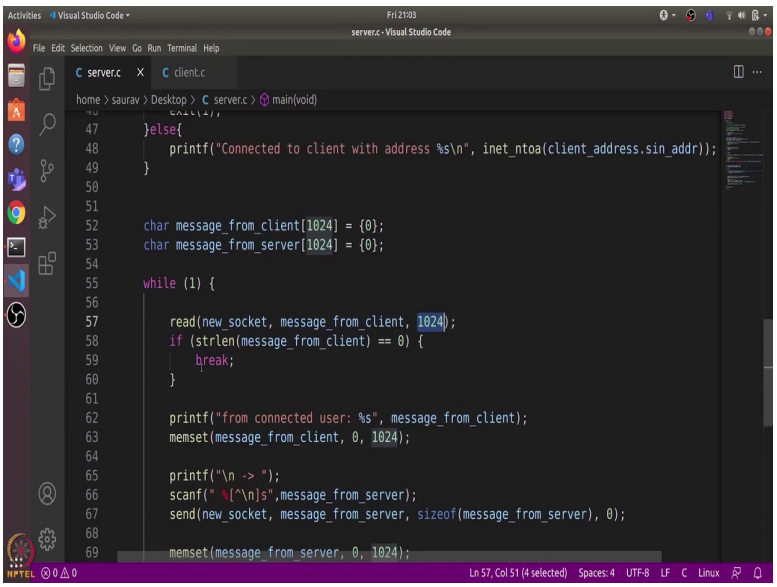

Activities
Visual Studio Code
Fri 21:02
server.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > server.c > main(void)
}else{
printf("Connected to client with address %s\n", inet_ntoa(client_address.sin_addr));
}
char message_from_client[1024] = {0};
char message_from_server[1024] = {0};
while (1) {
read(new_socket, message_from_client, 1024);
if (strlen(message_from_client) == 0) {
break;
}
printf("from connected user: %s", message_from_client);
memset(message_from_client, 0, 1024);
printf("\n -> ");
scanf(" %[^\n]s",message_from_server);
send(new_socket, message_from_server, sizeof(message_from_server), 0);
memset(message_from_server, 0, 1024);
Ln 53, Col 29 (19 selected)
Spaces: 4
UTF-8
LF
C
Linux

So, after we have successfully connected with the client, we will communicate with the client. So, we define two buffers. One is for the client messages, and another is for the server messages, then we have a while loop.

(Refer Slide Time: 4:25)

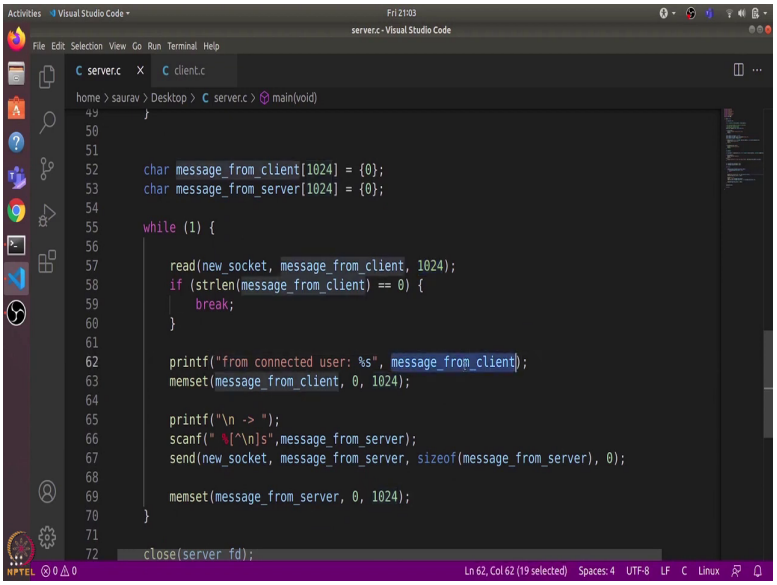


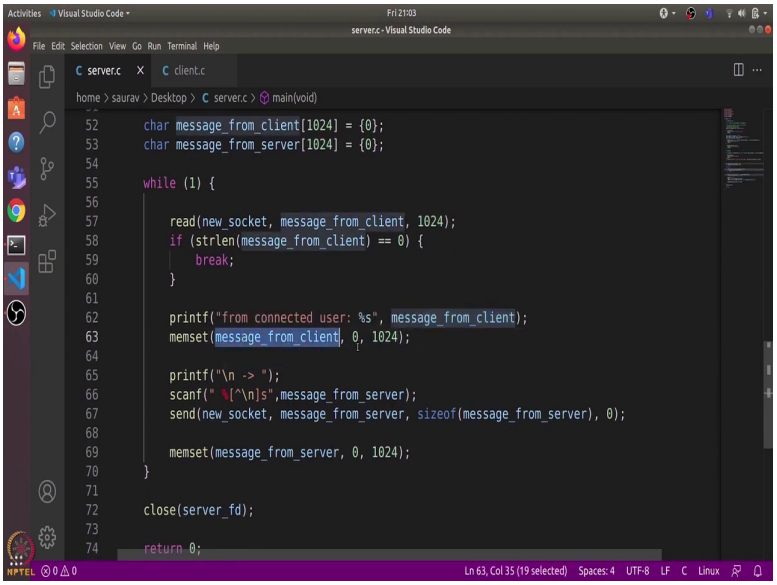


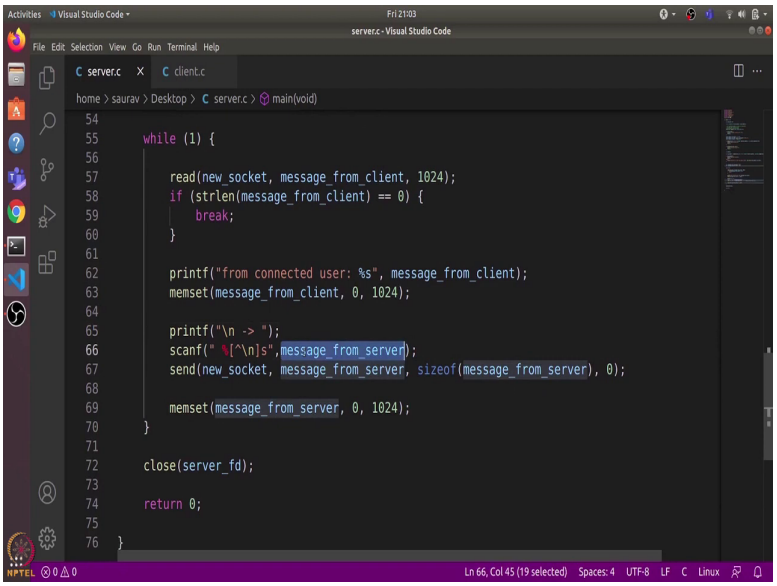


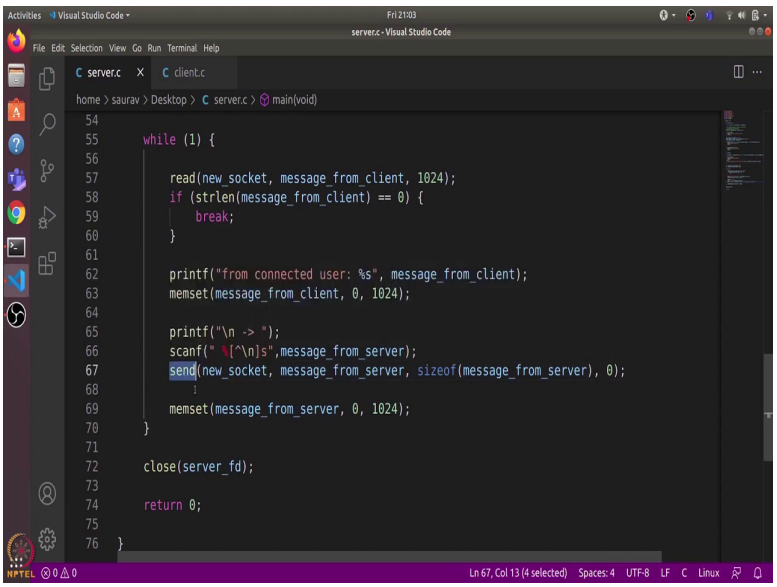
In the while loop first, we use the read system call to read the data which the client is sending in the location pointed to by this message from client address, and we read in 1024 bytes. Then we check if the message is empty then we break from this loop.

(Refer Slide Time: 4:41)



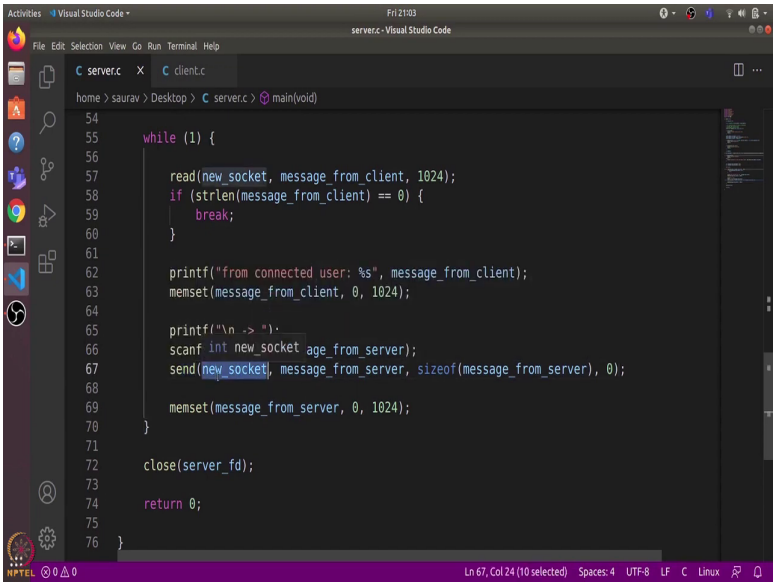

Activities
Visual Studio Code
Fri 21:03
server.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > server.c > main(void)
52 char message_from_client[1024] = {0};
53 char message_from_server[1024] = {0};
54
55 while (1) {
56
57 read(new_socket, message_from_client, 1024);
58 if (strlen(message_from_client) == 0) {
59 break;
60 }
61
62 printf("from connected user: %s", message_from_client);
63 memset(message_from_client, 0, 1024);
64
65 printf("\n -> ");
66 scanf(" %[^\n]s",message_from_server);
67 send(new_socket, message_from_server, sizeof(message_from_server), 0);
68
69 memset(message_from_server, 0, 1024);
70 }
71
72 close(server_fd);
73
74 return 0;
Ln 63, Col 35 (19 selected) Spaces: 4 UTF-8 LF C Linux

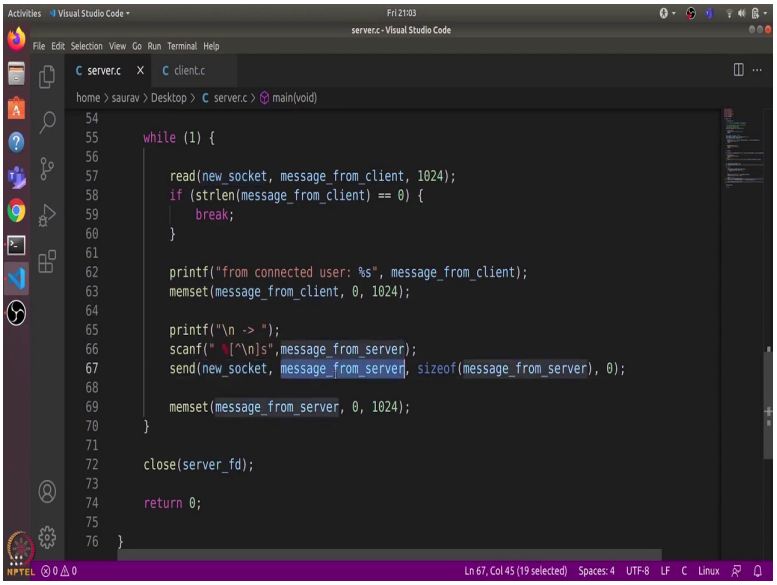

Activities
Visual Studio Code
Fri 21:03
server.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > server.c > main(void)
54
55 while (1) {
56
57 read(new_socket, message_from_client, 1024);
58 if (strlen(message_from_client) == 0) {
59 break;
60 }
61
62 printf("from connected user: %s", message_from_client);
63 memset(message_from_client, 0, 1024);
64
65 printf("\n -> ");
66 scanf(" %[^\n]s",message_from_server);
67 send(new_socket, message_from_server, sizeof(message_from_server), 0);
68
69 memset(message_from_server, 0, 1024);
70 }
71
72 close(server_fd);
73
74 return 0;
75
76 }
Ln 66, Col 45 (19 selected) Spaces: 4 UTF-8 LF C Linux

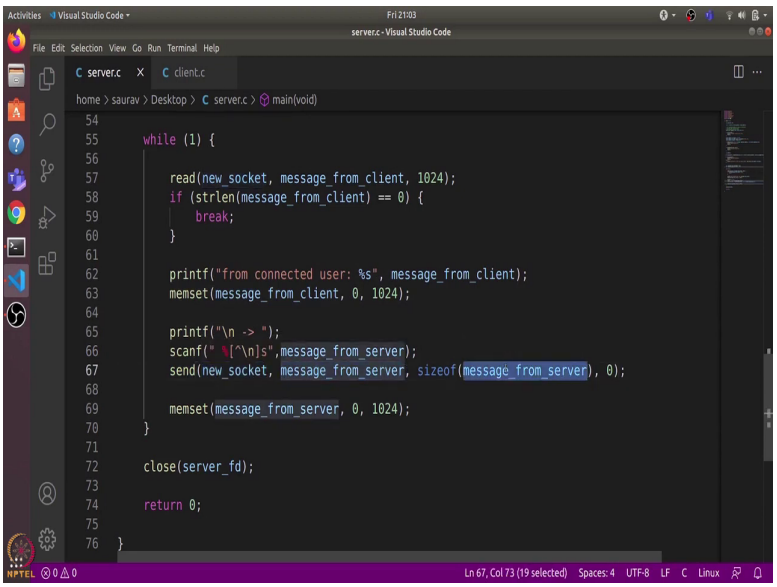

Activities
Visual Studio Code
Fri 21:03
server.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > server.c > main(void)
54
55 while (1) {
56
57 read(new_socket, message_from_client, 1024);
58 if (strlen(message_from_client) == 0) {
59 break;
60 }
61
62 printf("from connected user: %s", message_from_client);
63 memset(message_from_client, 0, 1024);
64
65 printf("\n -> ");
66 scanf(" %[^\n]s",message_from_server);
67 send(new_socket, message_from_server, sizeof(message_from_server), 0);
68
69 memset(message_from_server, 0, 1024);
70 }
71
72 close(server_fd);
73
74 return 0;
75
76 }
Ln 67, Col 13 (4 selected) Spaces: 4 UTF-8 LF C Linux

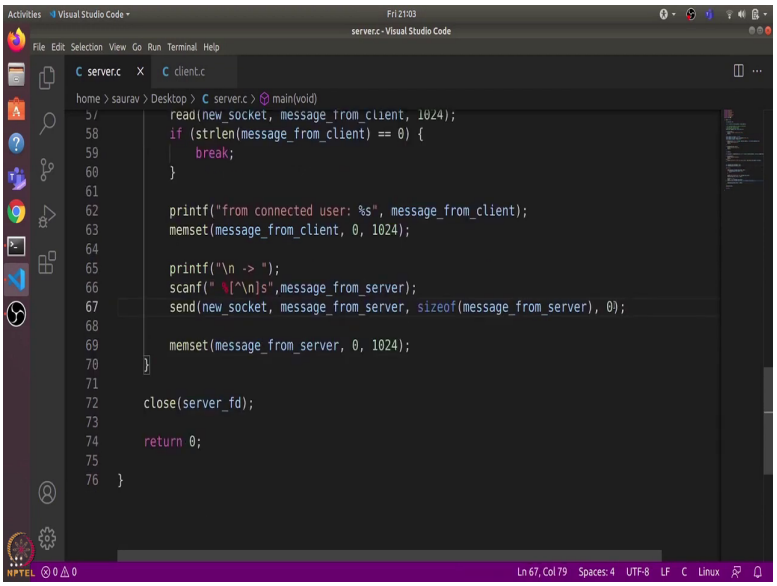
Otherwise, we print out the message that we received from the client and we reinitialize this message from client to all zeros. Then we print the sort of prompt so that the user can enter from the server side some message which then we can send to the client using the send system call.

(Refer Slide Time: 4:59)



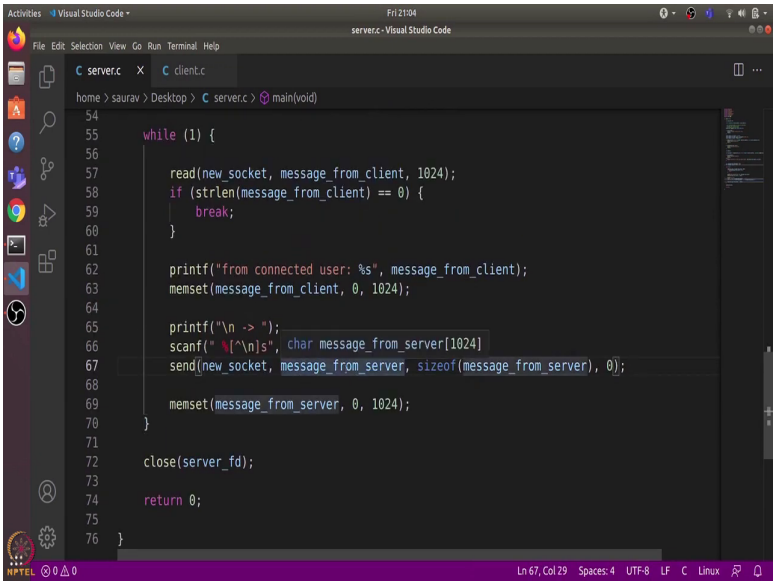


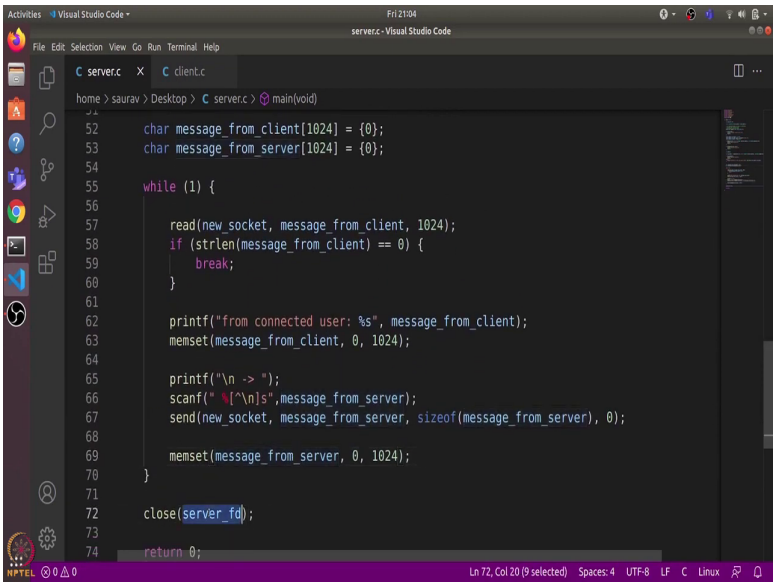


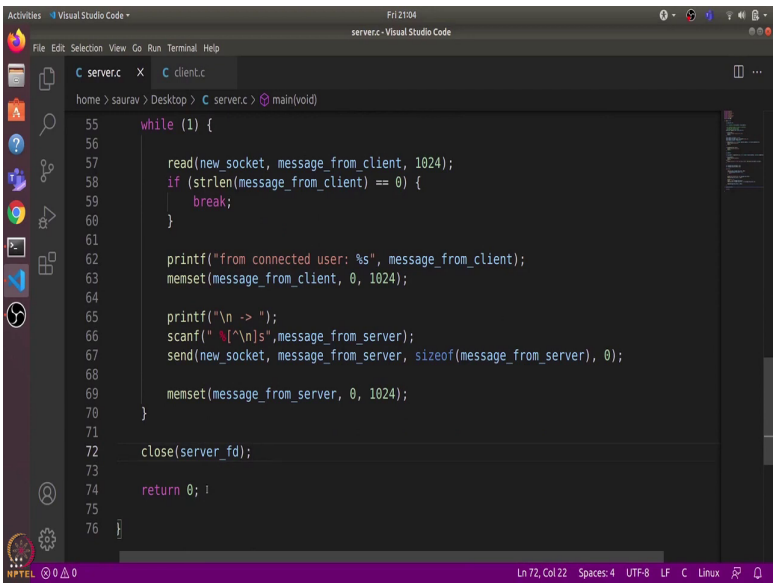


The send system call takes in the connected socket file descriptor, the message that you want to send and the size of the message. And the last argument represents the flags.

(Refer Slide Time: 5:09)

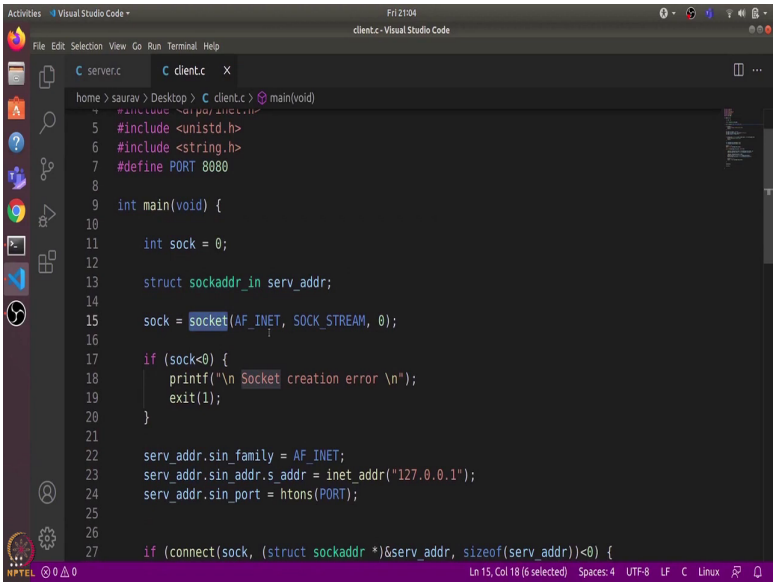

Activities
Visual Studio Code
Fri 21:04
server.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > server.c > main(void)
while (1) {
read(new_socket, message_from_client, 1024);
if (strlen(message_from_client) == 0) {
break;
}
printf("from connected user: %s", message_from_client);
memset(message_from_client, 0, 1024);
printf("\n -> ");
scanf(" %[^\n]s", char message_from_server[1024]
send(new_socket, message_from_server, sizeof(message_from_server), 0);
memset(message_from_server, 0, 1024);
}
close(server_fd);
return 0;
}
Ln 67, Col 29
Spaces: 4
UTF-8
LF
C
Linux

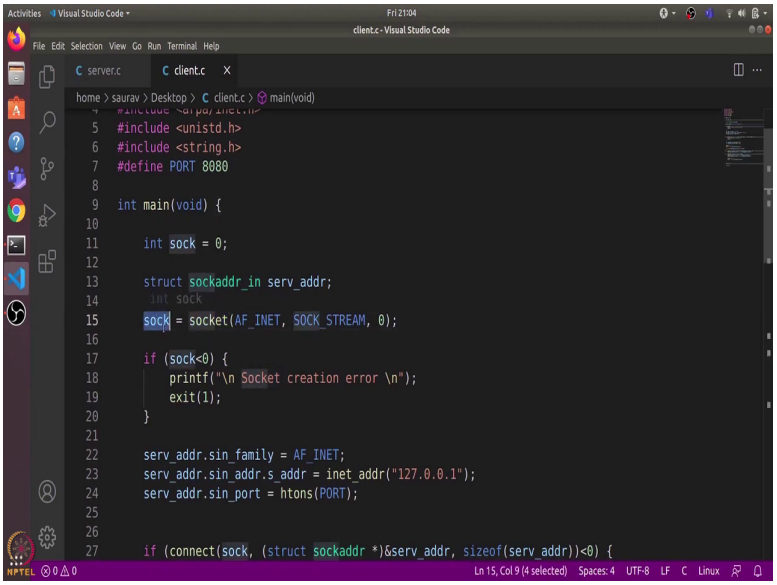

Activities
Visual Studio Code
Fri 21:04
server.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > server.c > main(void)
char message_from_client[1024] = {0};
char message_from_server[1024] = {0};
while (1) {
read(new_socket, message_from_client, 1024);
if (strlen(message_from_client) == 0) {
break;
}
printf("from connected user: %s", message_from_client);
memset(message_from_client, 0, 1024);
printf("\n -> ");
scanf(" %[^\n]s",message_from_server);
send(new_socket, message_from_server, sizeof(message_from_server), 0);
memset(message_from_server, 0, 1024);
}
close(server_fd);
return 0;
Ln 72, Col 20 (9 selected)
Spaces: 4
UTF-8
LF
C
Linux

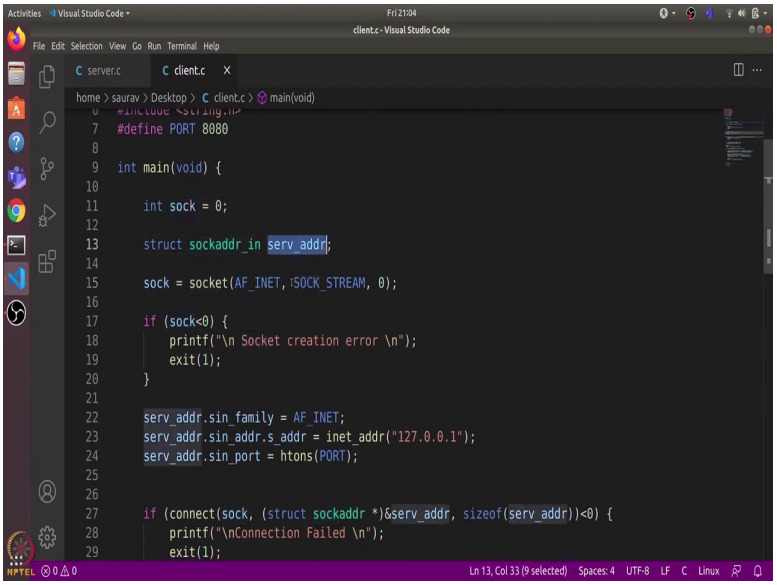

Activities
Visual Studio Code
Fri 21:04
server.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > server.c > main(void)
while (1) {
read(new_socket, message_from_client, 1024);
if (strlen(message_from_client) == 0) {
break;
}
printf("from connected user: %s", message_from_client);
memset(message_from_client, 0, 1024);
printf("\n -> ");
scanf(" %[^\n]s",message_from_server);
send(new_socket, message_from_server, sizeof(message_from_server), 0);
memset(message_from_server, 0, 1024);
}
close(server_fd);
return 0;
}
Ln 72, Col 22
Spaces: 4
UTF-8
LF
C
Linux

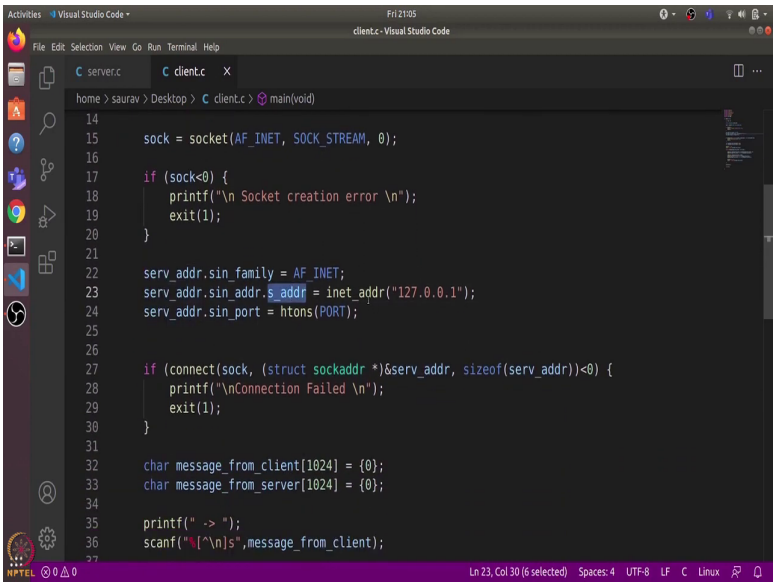
After sending this message to the client, we reset this message from server variable to all zeros. And this loop runs again and again. Finally, we close this server file descriptor and return from the program.

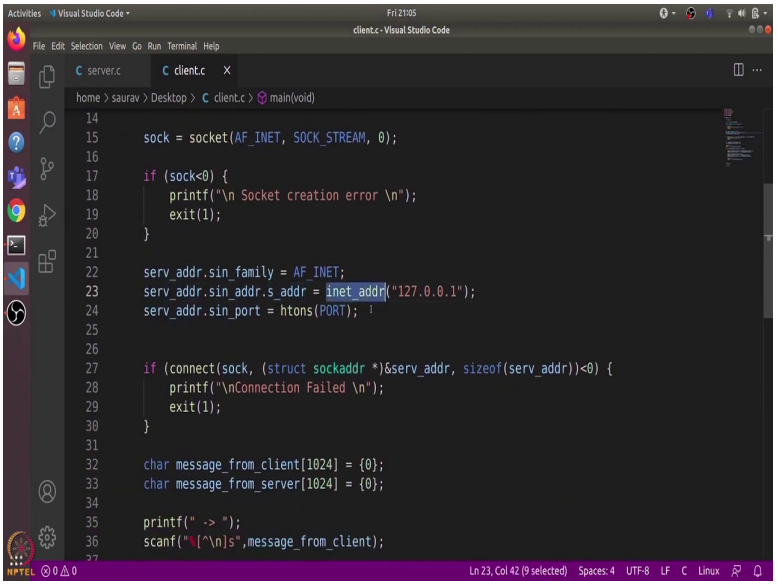
(Refer Slide Time: 5:25)





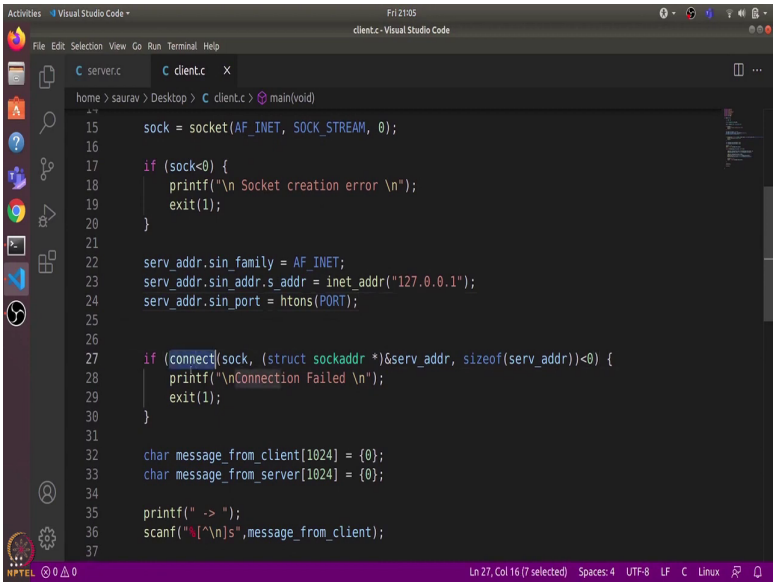

Activities
Visual Studio Code
Fri 21:04
client.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > client.c > main(void)
#define PORT 8080
int main(void) {
int sock = 0;
struct sockaddr_in serv_addr;
sock = socket(AF_INET, SOCK_STREAM, 0);
if (sock<0) {
printf("\n Socket creation error \n");
exit(1);
}
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
serv_addr.sin_port = htons(PORT);
if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr))<0) {
printf("\nConnection Failed \n");
exit(1);
Ln 13, Col 33 (9 selected) Spaces: 4 UTF-8 LF C Linux

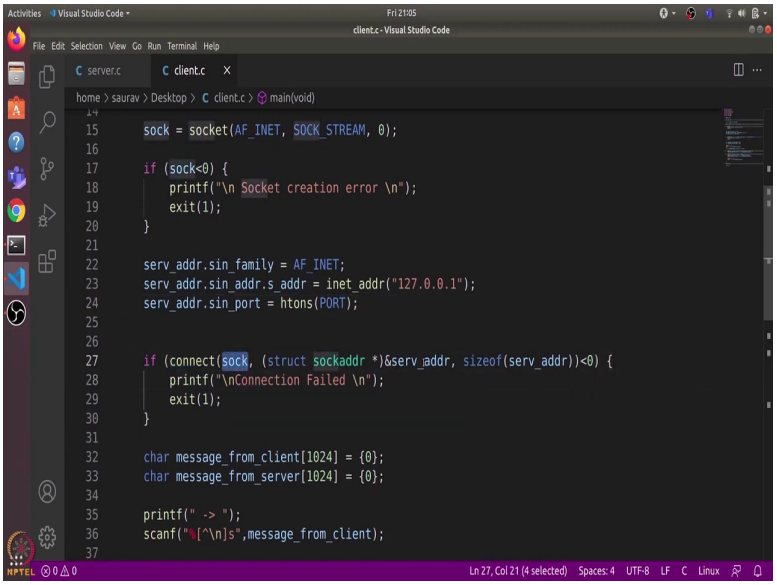

Activities
Visual Studio Code
Fri 21:05
client.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > client.c > main(void)
sock = socket(AF_INET, SOCK_STREAM, 0);
if (sock<0) {
printf("\n Socket creation error \n");
exit(1);
}
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
serv_addr.sin_port = htons(PORT);
if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr))<0) {
printf("\nConnection Failed \n");
exit(1);
}
char message_from_client[1024] = {0};
char message_from_server[1024] = {0};
printf(" -> ");
scanf("%[^\n]s",message_from_client);
Ln 23, Col 30 (6 selected) Spaces: 4 UTF-8 LF C Linux

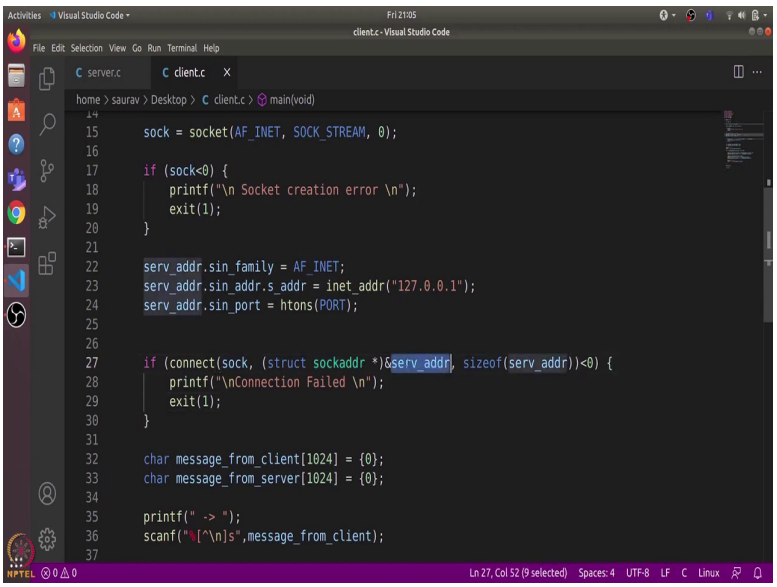

Activities
Visual Studio Code
Fri 21:05
client.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > client.c > main(void)
sock = socket(AF_INET, SOCK_STREAM, 0);
if (sock<0) {
printf("\n Socket creation error \n");
exit(1);
}
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
serv_addr.sin_port = htons(PORT);
if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr))<0) {
printf("\nConnection Failed \n");
exit(1);
}
char message_from_client[1024] = {0};
char message_from_server[1024] = {0};
printf(" -> ");
scanf("%[^\n]s",message_from_client);
Ln 23, Col 42 (9 selected) Spaces: 4 UTF-8 LF C Linux

So, that is the server.c program. Let us have a look at the client.c. In the client.c program again, we use the socket system call to create a new socket. And here we do not need to bind this socket to any address. But we need the server's address so that this socket can connect to the server. So, we define this socket at a struct and name it server address. And here we add the servers IP address and the port.

(Refer Slide Time: 5:54)

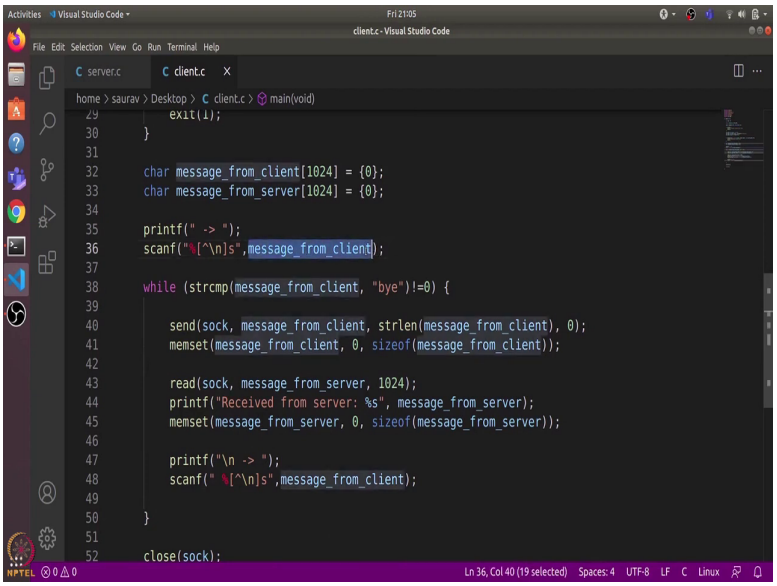


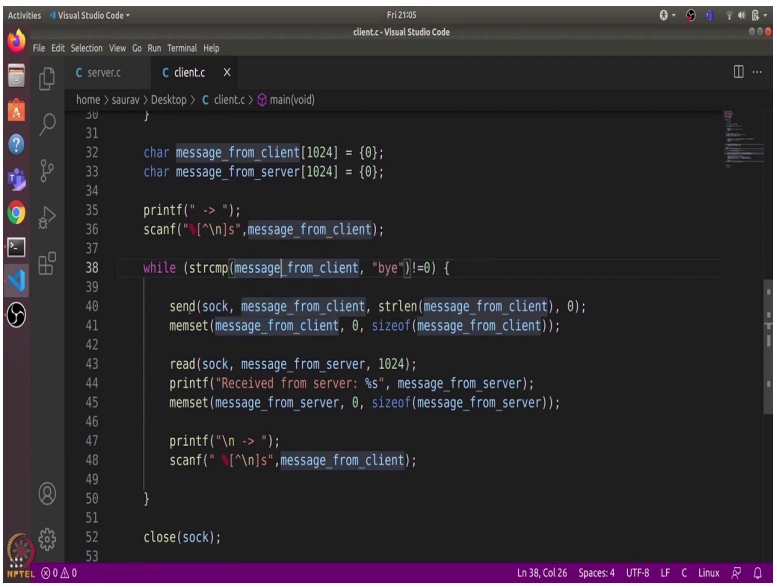


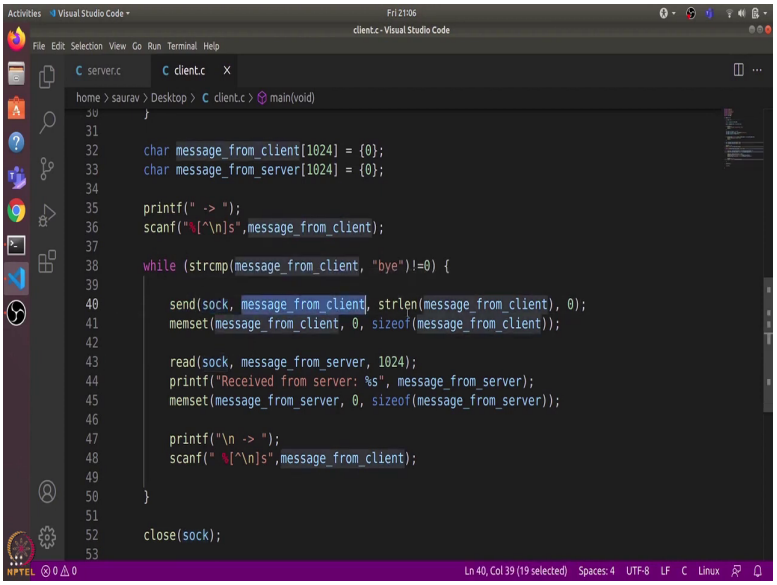


Then we use this connect system call to connect this particular client socket with the server, which is given by the IP address and the port. And here, the third argument is the size of this server address struct. And if it returns a negative value, which means there is some error, otherwise, the socket is successfully connected to the server.

(Refer Slide Time: 6:14)

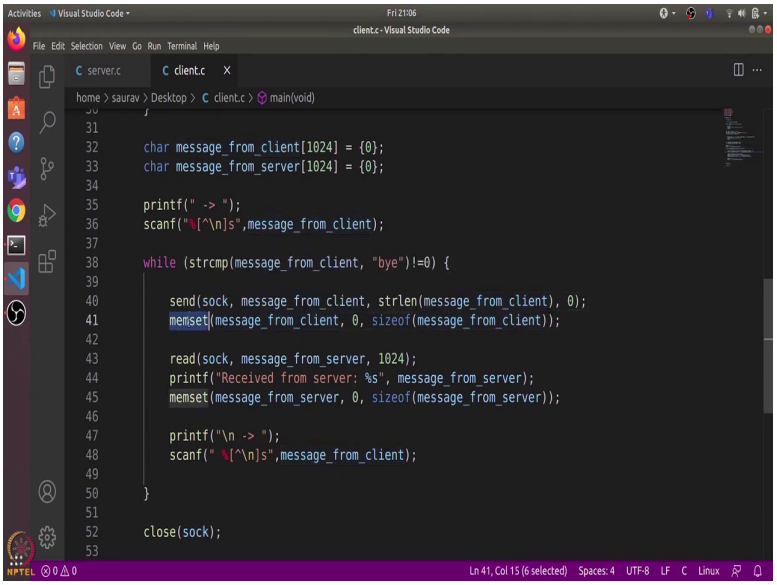


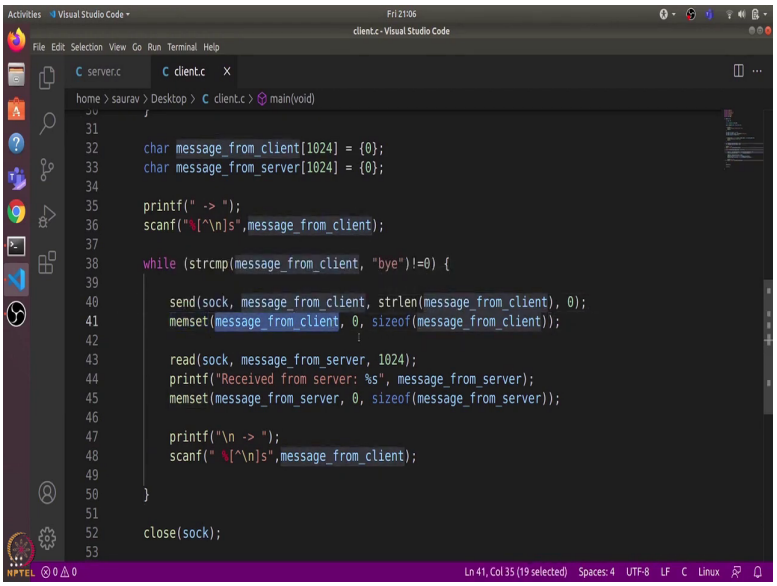


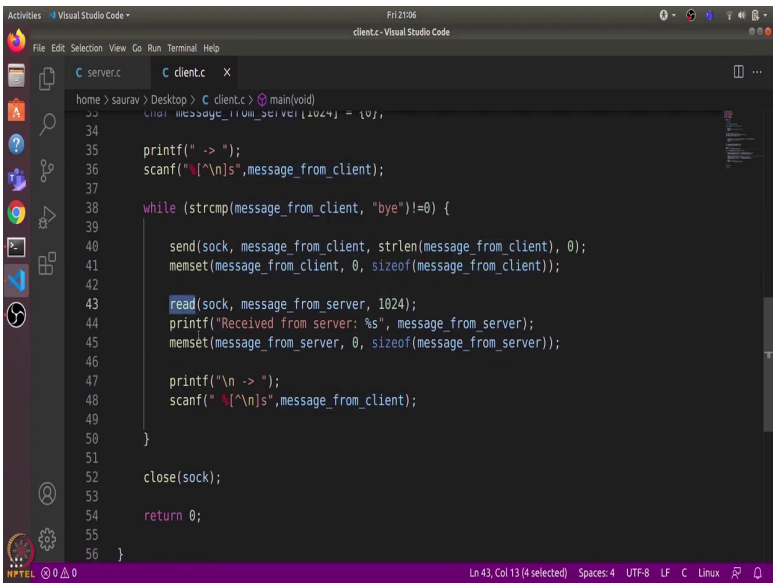


Now here also, we defined two buffers, one for client another for server. And we take in input from the user on the client side and store it in message from client. We ran a loop until this message from client is not equal to “bye” and send this message to the server using this send system call and this send system call takes in the socket file descriptor, the message from client and the length of the message. And finally, if we want to mention any flags.

(Refer Slide Time: 6:42)

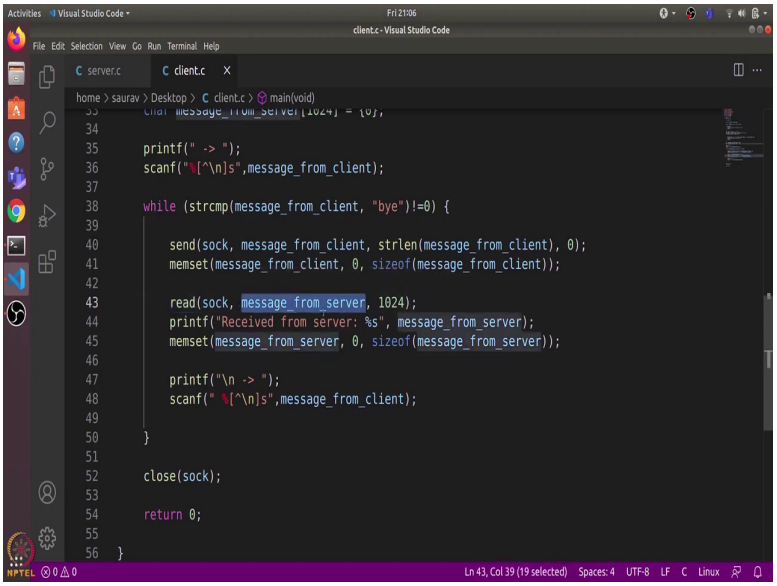

Activities
Visual Studio Code
Fri 21:06
client.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > client.c > main(void)
char message_from_client[1024] = {0};
char message_from_server[1024] = {0};
printf(" -> ");
scanf("%[^\n]s",message_from_client);
while (strcmp(message_from_client, "bye")!=0) {
send(sock, message_from_client, strlen(message_from_client), 0);
memset(message_from_client, 0, sizeof(message_from_client));
read(sock, message_from_server, 1024);
printf("Received from server: %s", message_from_server);
memset(message_from_server, 0, sizeof(message_from_server));
printf("\n -> ");
scanf(" %[^\n]s",message_from_client);
}
close(sock);
Ln 41, Col 15 (6 selected)
Spaces: 4
UTF-8
LF
C
Linux

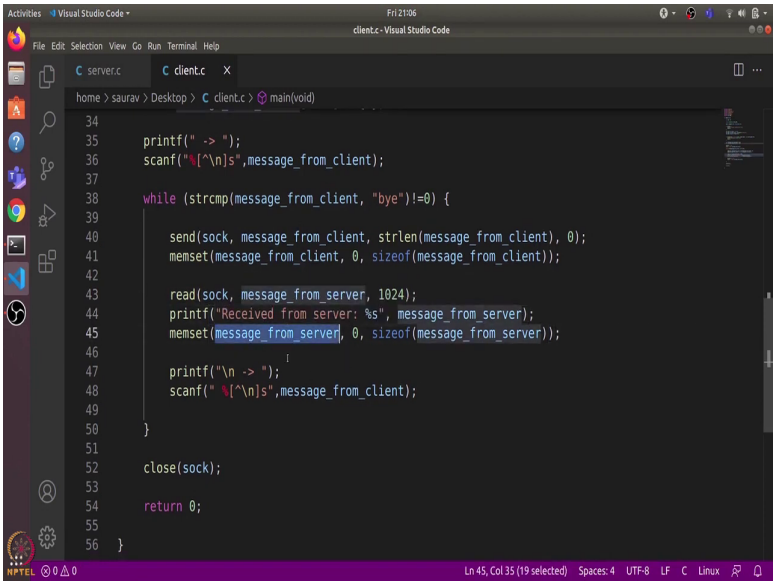

Activities
Visual Studio Code
Fri 21:06
client.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > client.c > main(void)
char message_from_client[1024] = {0};
char message_from_server[1024] = {0};
printf(" -> ");
scanf("%[^\n]s",message_from_client);
while (strcmp(message_from_client, "bye")!=0) {
send(sock, message_from_client, strlen(message_from_client), 0);
memset(message_from_client, 0, sizeof(message_from_client));
read(sock, message_from_server, 1024);
printf("Received from server: %s", message_from_server);
memset(message_from_server, 0, sizeof(message_from_server));
printf("\n -> ");
scanf(" %[^\n]s",message_from_client);
}
close(sock);
Ln 41, Col 35 (19 selected)
Spaces: 4
UTF-8
LF
C
Linux

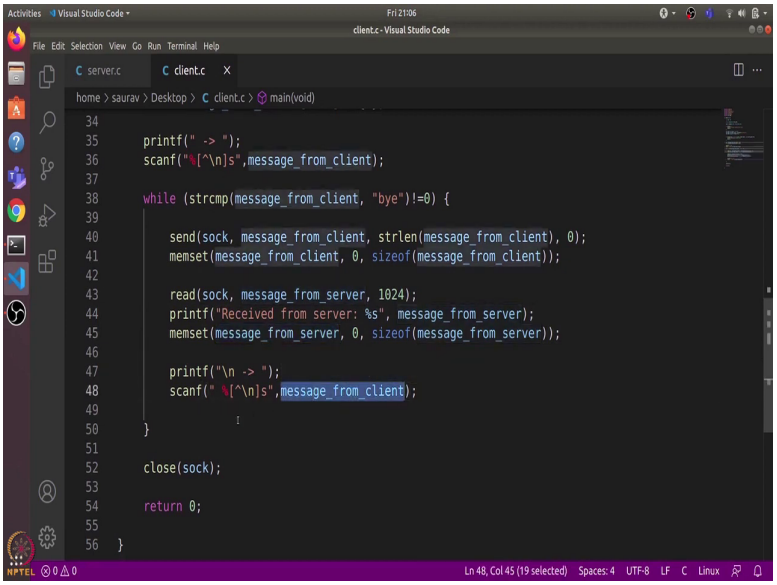

Activities
Visual Studio Code
Fri 21:06
client.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > client.c > main(void)
printf(" -> ");
scanf("%[^\n]s",message_from_client);
while (strcmp(message_from_client, "bye")!=0) {
send(sock, message_from_client, strlen(message_from_client), 0);
memset(message_from_client, 0, sizeof(message_from_client));
read(sock, message_from_server, 1024);
printf("Received from server: %s", message_from_server);
memset(message_from_server, 0, sizeof(message_from_server));
printf("\n -> ");
scanf(" %[^\n]s",message_from_client);
}
close(sock);
return 0;
}
Ln 43, Col 13 (4 selected)
Spaces: 4
UTF-8
LF
C
Linux

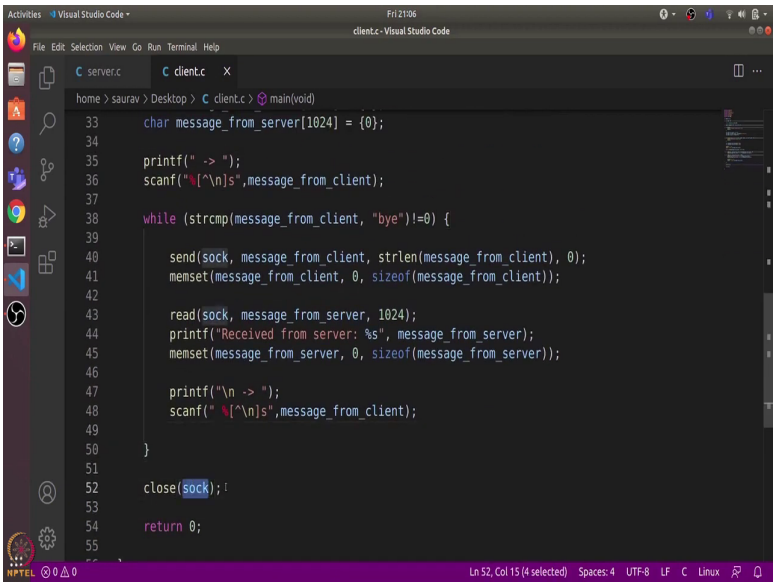
Then we reset this message from client to all zeros. And then we use the read system call to read the reply which comes from the server.

(Refer Slide Time: 6:52)



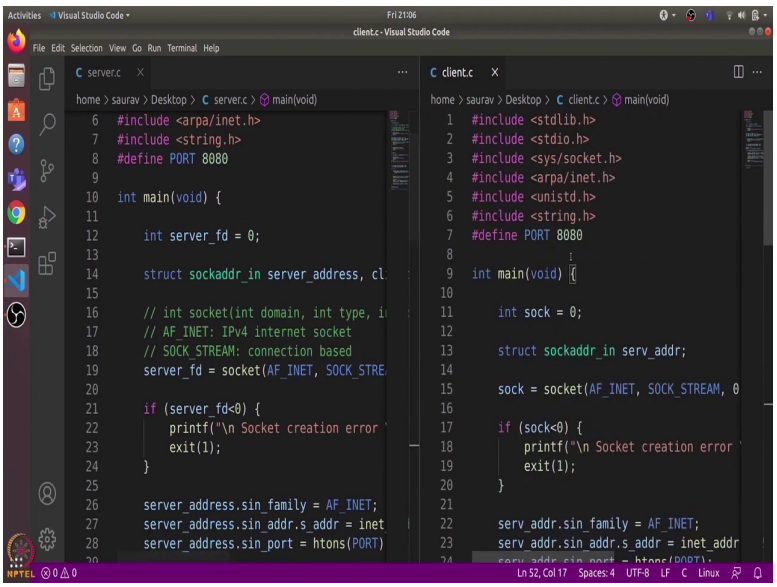


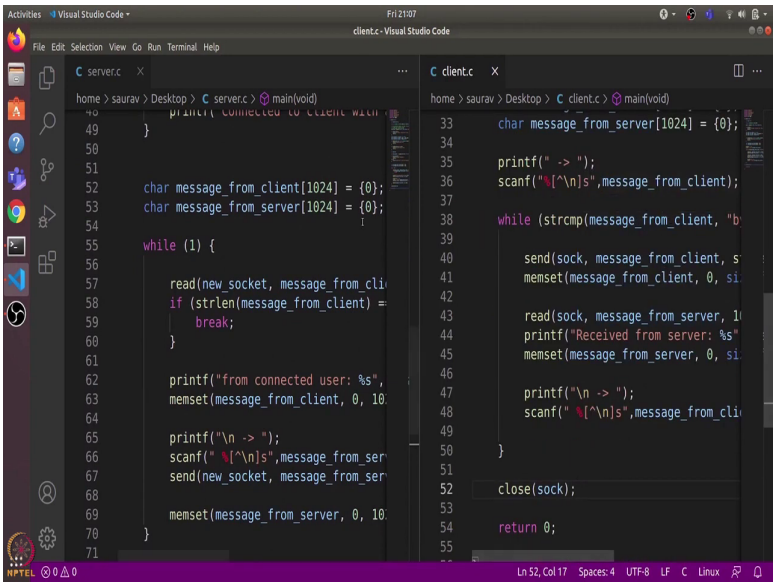


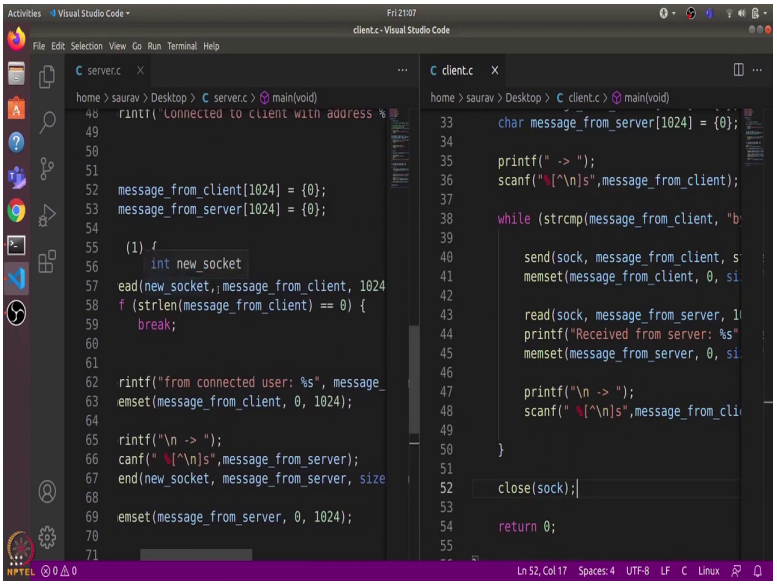


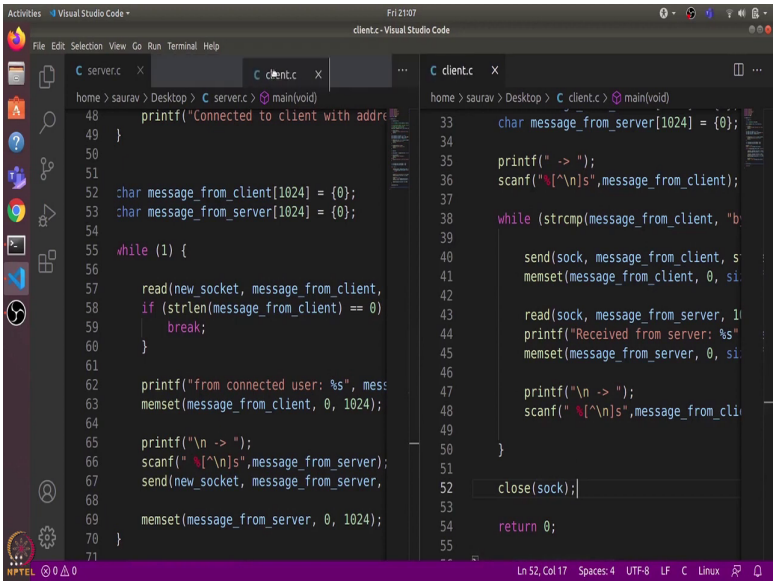
So, we read the that reply in the message from server. And we print out that we have received this message from the server. And we reset this message from server variable. And again, we take in message from client from the user. And this whole thing repeats in a while loop. Finally, we close the socket using the closed system call.

(Refer Slide Time: 7:12)


Activities
Visual Studio Code
Fri 21:06
client.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > server.c > main(void)
home > saurav > Desktop > client.c > main(void)
#include <arpa/inet.h>
#include <string.h>
#define PORT 8080
int main(void) {
int server_fd = 0;
struct sockaddr_in server_address, cl
// int socket(int domain, int type, i
// AF_INET: IPv4 internet socket
// SOCK_STREAM: connection based
server_fd = socket(AF_INET, SOCK_STRE
if (server_fd<0) {
printf("\n Socket creation error
exit(1);
}
server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = inet
server_address.sin_port = htons(PORT)
#include <stdlib.h>
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#define PORT 8080
int main(void) {
int sock = 0;
struct sockaddr_in serv_addr;
sock = socket(AF_INET, SOCK_STREAM, 0
if (sock<0) {
printf("\n Socket creation error
exit(1);
}
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr
Ln 52, Col 17
Spaces: 4
UTF-8
LF
C
Linux

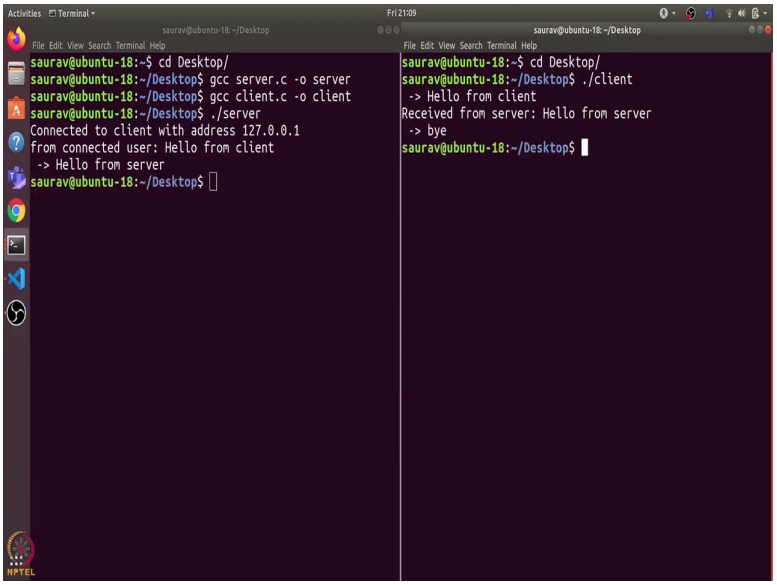

Activities
Visual Studio Code
Fri 21:07
client.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > server.c > main(void)
home > saurav > Desktop > client.c > main(void)
}
char message_from_client[1024] = {0};
char message_from_server[1024] = {0};
while (1) {
read(new_socket, message_from_cli
if (strlen(message_from_client) =
break;
}
printf("from connected user: %s",
memset(message_from_client, 0, 10
printf("\n -> ");
scanf(" %[^\n]s",message_from_ser
send(new_socket, message_from_ser
memset(message_from_server, 0, 10
}
char message_from_server[1024] = {0};
printf(" -> ");
scanf("%[^\n]s",message_from_client);
while (strcmp(message_from_client, "b
send(sock, message_from_client, s
memset(message_from_client, 0, si
read(sock, message_from_server, 1
printf("Received from server: %s"
memset(message_from_server, 0, si
printf("\n -> ");
scanf(" %[^\n]s",message_from_cli
}
close(sock);
return 0;
Ln 52, Col 17
Spaces: 4
UTF-8
LF
C
Linux


Activities
Visual Studio Code
Fri 21:07
client.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
server.c
client.c
home > saurav > Desktop > server.c > main(void)
home > saurav > Desktop > client.c > main(void)
rintf("Connected to client with address %
message_from_client[1024] = {0};
message_from_server[1024] = {0};
(1) {
int new_socket
ead(new_socket,message_from_client, 1024
f (strlen(message_from_client) == 0) {
break;
rintf("from connected user: %s", message_
emset(message_from_client, 0, 1024);
rintf("\n -> ");
canf(" %[^\n]s",message_from_server);
end(new_socket, message_from_server, size
emset(message_from_server, 0, 1024);
char message_from_server[1024] = {0};
printf(" -> ");
scanf("%[^\n]s",message_from_client);
while (strcmp(message_from_client, "b
send(sock, message_from_client, s
memset(message_from_client, 0, si
read(sock, message_from_server, 1
printf("Received from server: %s"
memset(message_from_server, 0, si
printf("\n -> ");
scanf(" %[^\n]s",message_from_cli
}
close(sock);
return 0;
Ln 52, Col 17
Spaces: 4
UTF-8
LF
C
Linux



So, that is the overall client dot C program. Let us have a look at them side by side. So, what will happen is the server and client will establish connected sockets. And once they have connected sockets, then client will send a message to the Server and Server will use this read system called to read that message. It will print out that message and then it will send some message from the server side. And then the client reads that message. And again, it sends some other message to the server. And that is how they communicate with each other. So, let us compile and turn these programs.

(Refer Slide Time: 7:47)



So, I will open a terminal. And let us increase the font size. And I will open one more terminal so that we have one terminal for server and another for client. So, let us go to the desktop. And I will compile server.c and I will name the executable as server and also compile client.c and name this executable as client. Now what I will do is I will first run the server and then I will run the client so that it can establish connections.

So, once I run the server, now the server is running and it is listening for incoming connections. Now when I run client here, it uses that connect system call to connect with the server. And we have this connected to client with address 127.0.0.1 which is just a localhost IP address on the server side. So, let us send some message to the server. Hello from client. So, you can see that it shows up here on the server side that there is a message sent from the client. And let us reply back hello from server and we received from the server on this side.

So, that is how two processes can communicate with each other. So, here we have used localhost IP address. But in general, if the processes are on different machines, then they will have different IP addresses and a specific port using which the server and client can connect to each other. So, let us type in bye and that will end the connection. So, that is it for this video. Thanks and have a nice day.