**SSN College of Engineering, Kalavakkam**

**Department of Computer Science and Engineering**

**V Semester - CSE 'B'**

**UCS1511 NETWORKS LAB**

**Date :**13/08/2020 **Name :** Rahul Ram M

**Exercise :** 03 **Reg No :** 185001121

**CHAT USING TCP**

**Learning Objective:**

To write a socket program to perform chat with multiple clients.

**Algorithm for Server:**

1. Creating a socket using the function socket(domain, type, protocol) which the returns an integer as the status of the socket creation. Here the domain is AF\_INET(iPv4 protocol), type is SOCK\_STEAM and protocol as 0.
2. Using bzero(&server\_addr, sizeof(server\_addr)) function setting values of all the socket structures to null.
3. Using bind() to binf the socket to the address and port number specified in addr(custom data structure). Here, we bind the server to the localhost, hence we use INADDR\_ANY to specify the IP address.
4. listen() function is used to set the server socket in the passive mode, where it waits for the client to approach the server to make a connection, with maximum number of connection in this case is 3.
5. Intitialize all the values of the client\_socket array to 0(means we don’t have to listen to them).
6. Setting a while loop which runs till server enters ‘exit’ or terminate using ctrl+z.

* Clear the list of socket descriptors to monitor using FD\_ZERO(&read\_fds).
* Add the descriptor of the server to the list using FD\_SET(server\_socket, &read\_fds).
* Assign max\_sd as server\_socket.
* Using for loop over client\_socket array to select valid descriptors , add them to the list of descriptors to monitor and assign the higher number to max\_fd.
* Using select() wait for the activity on one of the sockets in the read\_fds indefinetely(timeout is NULL).
* Using FD\_ISSET(server\_socket, &read\_fds) tests for an incoming connection.
  + If it detects any incoming connection, accept the connection using accept() which creates a socket and assign the new socket any free space in client\_socket array.
* Now loop over all the valid fds in the client\_socket array
  + Using FD\_ISSET(sd, &read\_fds) tests for any message from the descriptor.
  + If the value read by read() on this descriptor is 0, close the socket using close() and reassign the value in the array to be 0(Terminates the connection to that client) for reuse.
  + Else read() has read some message sent by the client to the server.
  + Print the message.
  + Read message from server using scanf(" %[^\n]", buffer) and write to the client using ‘write(sd, buffer, sizeof(buffer))’ if the message is not ‘exit’, else terminate the program.

**Algorithm for Client(same for all the clients):**

1. Creating a socket using the function socket(domain, type, protocol) which the returns an integer as the status of the socket creation. Here the domain is AF\_INET(iPv4 protocol), type is SOCK\_STEAM and protocol as 0.
2. Using bzero(&server\_addr, sizeof(server\_addr)) function setting values of all the socket structures to null.
3. The above two steps are same as the server.
4. The connect() system call connects the socket referred to by the file descriptor socket\_fd to the address specified by server\_addr. Server’s address and port is specified in server\_addr.
5. Read the client name from the client using scanf()
6. Setting a while loop which runs till ‘exit’ is given as message.
   1. Clearing the buffer using bzero().
   2. Reading message from the client using ‘scanf(" %[^\n]", message)’ and concatenate name of the client , ‘:’ symbol and message and save it in buffer.
   3. If the message is ‘exit’, close the descriptor using close() and break the loop.
   4. Else write the message in the buffer to the server using write(client\_fd, buff, sizeof(buff)).
   5. Now read the message from the server using ‘value = read(client\_fd, buff, sizeof(buff))’ and print the message from the server using printf();

**Program for Server:**

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <errno.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <sys/time.h>

#define TRUE 1

#define FALSE 0

#define PORT 8080

int main()

{

int server\_socket, addr\_size, new\_socket, client\_socket[10],

max\_clients = 10, activity, val, sd, max\_sd, opt = TRUE;

struct sockaddr\_in server\_addr;

char buffer[1024];

fd\_set read\_fds;

for (int i = 0; i < max\_clients; i++)

{

client\_socket[i] = 0;

}

if((server\_socket = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

perror("Socker error");

exit(1);

}

if( setsockopt(server\_socket, SOL\_SOCKET, SO\_REUSEADDR, (char \*)&opt, sizeof(opt)) < 0 )

{

perror("setsockopt");

exit(1);

}

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

server\_addr.sin\_port = htons(PORT);

if(bind(server\_socket, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0)

{

perror("Bind error: ");

exit(1);

}

if(listen(server\_socket,3) < 0)

{

perror("Listen error");

exit(1);

}

addr\_size = sizeof(server\_addr);

printf("Waiting for client...\n");

while(TRUE)

{

FD\_ZERO(&read\_fds);

FD\_SET(server\_socket, &read\_fds);

max\_sd = server\_socket;

for(int i = 0; i < max\_clients; i++)

{

sd = client\_socket[i];

if(sd > 0)

{

FD\_SET(sd, &read\_fds);

}

if(sd > max\_sd)

{

max\_sd = sd;

}

}

activity = select( max\_sd + 1, &read\_fds, NULL, NULL, NULL);

if ((activity < 0) && (errno!=EINTR))

{

printf("Select error");

}

if (FD\_ISSET(server\_socket, &read\_fds))

{

if ((new\_socket = accept(server\_socket, (struct sockaddr \*)&server\_addr, (socklen\_t\*)&addr\_size))<0)

{

perror("Accept error");

exit(1);

}

for (int i = 0; i < max\_clients; i++)

{

if( client\_socket[i] == 0 )

{

client\_socket[i] = new\_socket;

break;

}

}

}

for(int i = 0; i < max\_clients; i++)

{

sd = client\_socket[i];

if (FD\_ISSET(sd, &read\_fds))

{

if ((val = read(sd , buffer, sizeof(buffer)) == 0))

{

close(sd);

client\_socket[i] = 0;

}

else

{

printf("%s\n", buffer);

bzero(buffer, sizeof(buffer));

printf("Server : ");

scanf(" %[^\n]", buffer);

if(strcmp(buffer, "exit") == 0)

{

printf("Server shutdown successfully!\n");

exit(0);

}

val = write(sd, buffer, sizeof(buffer));

}

}

}

}

return 0;

}

**Program for Client(same for all the clients):**

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define PORT 8080

int main()

{

int client\_fd, value;

struct sockaddr\_in server\_addr;

char buff[1024], name[10], message[1000];

printf("Connecting to server...\n");

if((client\_fd=socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

perror("Socket error");

}

bzero(&server\_addr,sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

server\_addr.sin\_port = htons(PORT);

if(connect(client\_fd,(struct sockaddr\*)&server\_addr, sizeof(server\_addr)) != 0)

{

perror("Connect error");

}

else

{

printf("Connected to the Server...\n");

}

printf("Enter client name : ");

scanf("%s", name);

while(1)

{

bzero(buff, 1024);

printf("%s : ", name);

scanf(" %[^\n]", message);

strcat(buff, name);

strcat(buff, " : ");

strcat(buff, message);

if(strcmp(message, "exit") == 0){

close(client\_fd);

printf("Disconnected from server...\n");

break;

}

else

{

value = write(client\_fd, buff, sizeof(buff));

}

value = read(client\_fd, buff, sizeof(buff));

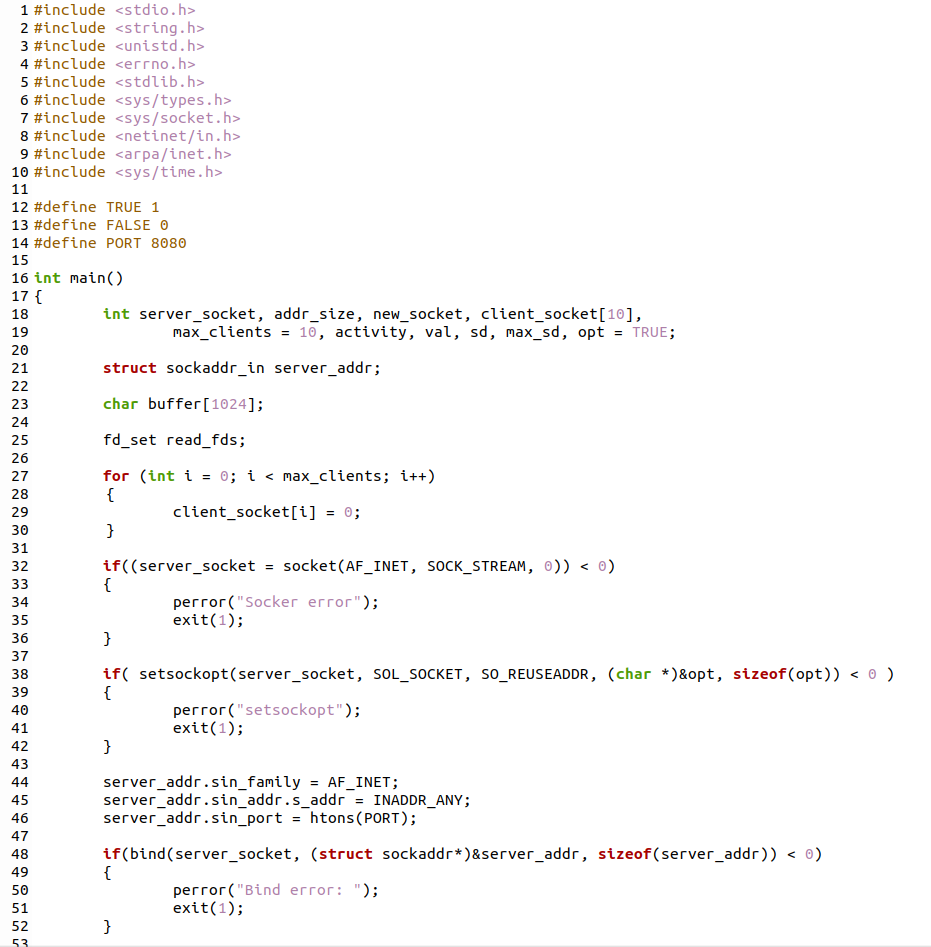
printf("Server: %s\n", buff);

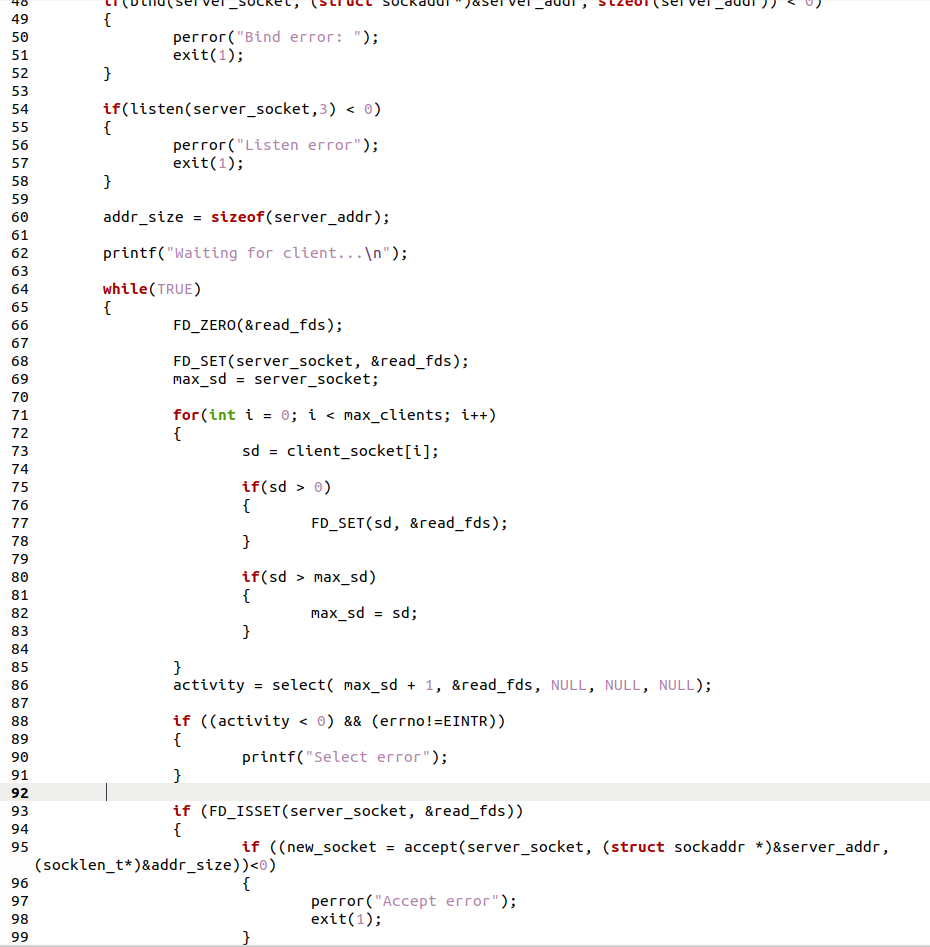
}

return 0;

}

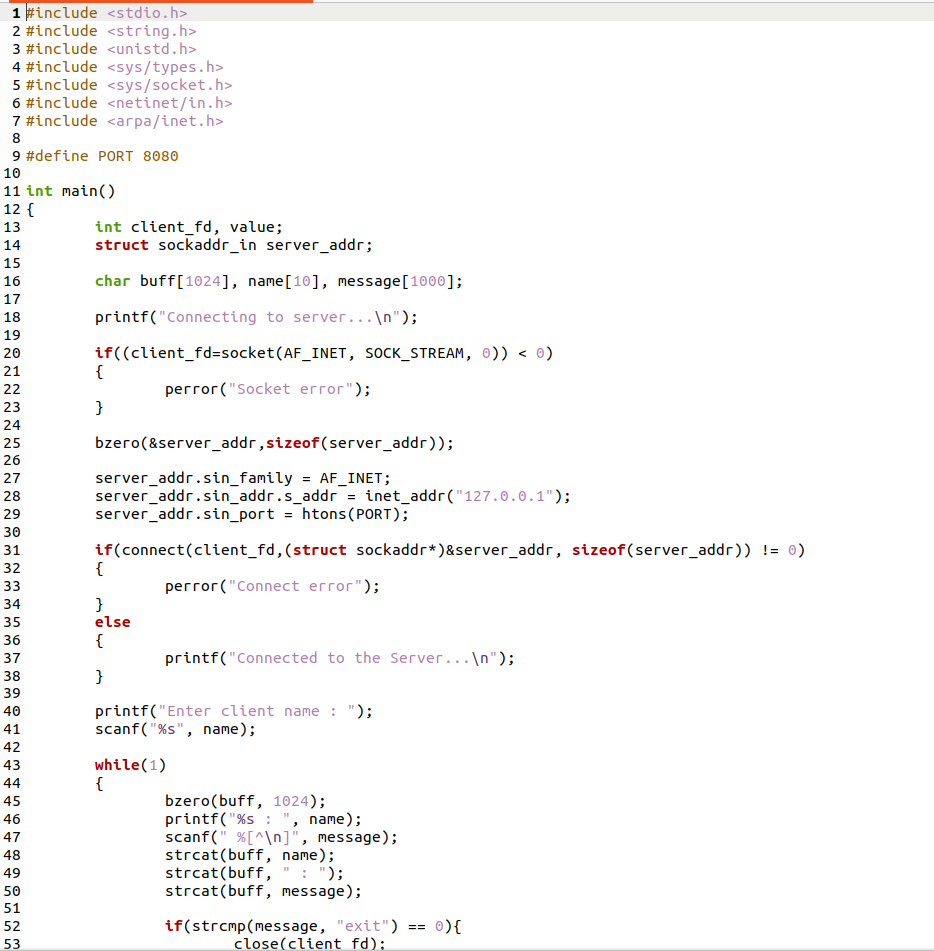
**Screenshot for Server:**

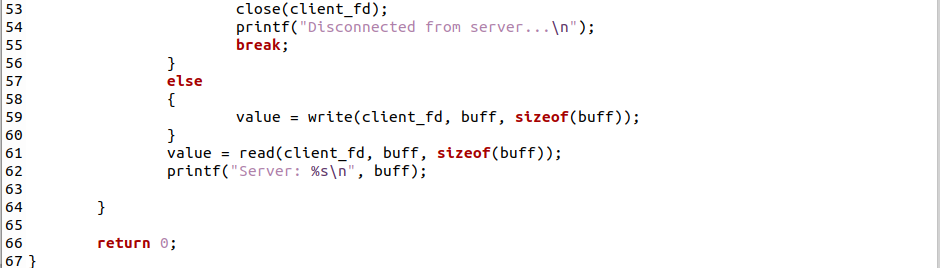
****

****

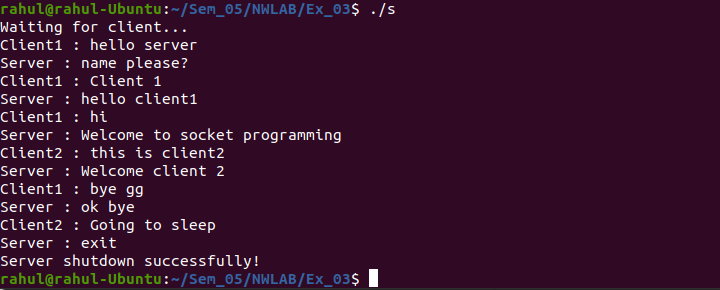
****

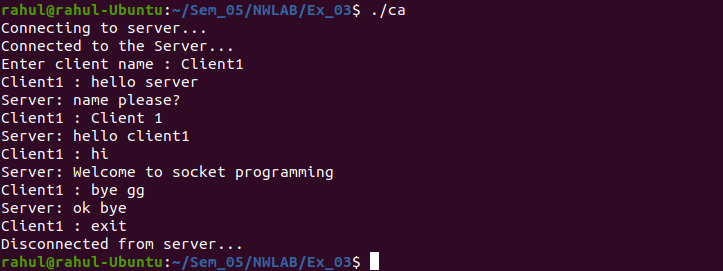
**Screenshot for Client:**

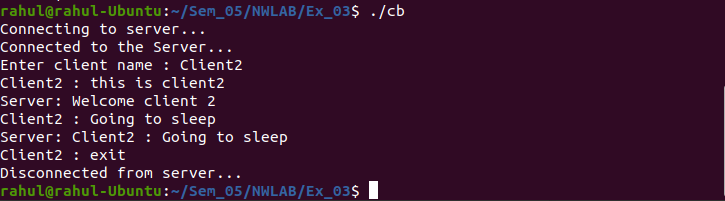
****

****

**Server Output:**

**Client1 Output:**

**Client2 Output:**

****

**Learning Outcomes:**

This assignment helped me to

1. Write program for server and client with socket programming.
2. Understand various functions invloved in creating, estabilishing, maintaining, Sending, recieving and termininating the connection between the server and client.
3. Connect multiple clients to the server using select() system call.
4. Learn about the data structures used for select() :fd\_set.
5. Write code to make server and client communicate with each other using read() and write() functions.