Program No.12 MULTIUSER CHAT SERVER AND CLIENT

Aim:-

To Implement a multi user chat server using TCP as transport layer protocol.

Problem description – Using TCP socket, create connection between multiple clients and single server.

Algorithm – TCP SERVER

1. Create a socket for TCP using the function call, socket(AF_INET, SOCK_STREAM, 0);
2. The memset() function fills the first n bytes of memory area pointed to by addr with constant byte 0.
3. Initialize the structure sockaddr_in members of sin_family, sin_addr, sin_port
4. Bind the socket to its port using bind(int sockfd, (struct sockaddr *) &ser_addr, sizeof(ser_addr));
5. Listen for any active client connections using listen(int sockfd, int backlog); Backlog argument defines the maximum length to which queue of pending connections for sockfd may grow.
6. Server infinitely accepts client connections using accept function call as follows: accept(int sockfd, (struct sockaddr *) &cl_addr, &sizeof(cl_addr));
7. After accepting client connection, inet_ntop() function is used to convert clients network address structure src in the af address family into a character string. The resulting string is copied to the buffer pointed to by dst, which must be a non-null pointer. The caller specifies the number of bytes available in this buffer in argument size.
   #include <arpa/inet.h>
   const char *inet_ntop(int af, const void *src, char *dst, socklen_t size);
8. Child process is created. Parent process stops listening for new connections. Child will continue to listen. The main (parent) process now handles the connected client.
9. After clearing the buffer memory area using memset() function, data is received from client using recv(int sockfd, void *buffer, BUF_SIZE, unsigned int flags);
10. Sends back received data to client using send(int sockfd, void *buffer, BUF_SIZE, unsigned int flags) function
11. Prints to which client IP address data was sent.
12. Close the socket using close(int sockfd) function.

Algorithm – TCP CLIENT

1. Create a socket for TCP using the function call, socket(AF_INET, SOCK_STREAM, 0);
2. The memset() function fills the first n bytes of memory area pointed to by addr with constant byte 0.
3. Initialize the structure sockaddr_in members of sin_family, sin_addr, sin_port
4. Connect using function connect(int sockfd, (struct sockaddr *) &ser_addr, sizeof(ser_addr));
5. Client reads in the line and make sure it was successful by processing the line using fgets() function infinitely in a while loop as follows: while(fgets(buffer, BUF_SIZE, stdin) != NULL))
6. Client sends data to server using send(int sockfd, void *buffer, BUF_SIZE, unsigned int flags) function
7. Client receives response from server using using recv() function as follows: recv(int sockfd, void *buffer, BUF_SIZE, unsigned int flags);
8. Prints the received message in client's terminal.
9. Client can continue sending messages to server, as long as server is listening.

TCP MULTIUSER SERVER – multiuserserver.c

```
#include"stdio.h"
#include"stdlib.h"
#include"sys/types.h"
#include"sys/socket.h"
#include"string.h"
#include"netinet/in.h"
#define PORT 4444
#define BUF_SIZE 2000
```

```
#define CLADDR_LEN 100
void main() {
 struct sockaddr_in addr, cl_addr;
 int sockfd, len, ret, newsockfd;
 char buffer[BUF_SIZE];
 pid_t childpid;
 char clientAddr[CLADDR_LEN];
  sockfd = socket(AF_INET, SOCK_STREAM, 0);
 if (sockfd < 0) {
  printf("Error creating socket!\n");
  exit(1);           }
 printf("Socket created...\n");
 memset(&addr, 0, sizeof(addr));
 addr.sin_family = AF_INET;
 addr.sin_addr.s_addr = INADDR_ANY;
 addr.sin_port = PORT;
 ret = bind(sockfd, (struct sockaddr *) &addr, sizeof(addr));
 if (ret < 0) {
  printf("Error binding!\n");
  exit(1);     }
 printf("Binding done...\n");
 printf("Waiting for a connection...\n");
 listen(sockfd, 5);
 for (;;) { //infinite loop
  len = sizeof(cl_addr);
  newsockfd = accept(sockfd, (struct sockaddr *) &cl_addr, &len);
  if (newsockfd < 0) {
   printf("Error accepting connection!\n");
   exit(1);                }
  printf("Connection accepted...\n");
  inet_ntop(AF_INET, &(cl_addr.sin_addr), clientAddr, CLADDR_LEN);
  if ((childpid = fork()) == 0) { //creating a child process
 close(sockfd);
//stop listening for new connections by the main process.
//the child will continue to listen.
//the main process now handles the connected client.
   for (;;) {
    memset(buffer, 0, BUF_SIZE);
    ret = recv(newsockfd, buffer, BUF_SIZE, 0);
    if(ret < 0) {
     printf("Error receiving data!\n");
     exit(1);    }
    printf("Received data from %s: %s\n", clientAddr, buffer);
    ret = send(newsockfd, buffer, BUF_SIZE, 0);
    if (ret < 0)  {
     printf("Error sending data!\n");
     exit(1);      }
    printf("Sent data to %s: %s\n", clientAddr, buffer);
   }   }   close(newsockfd);
 } }
```

TCP MULTIUSER CLIENT – multiuserclient.c

```
#include"stdio.h"
#include"stdlib.h"
#include"sys/types.h"
#include"sys/socket.h"
#include"string.h"
#include"netinet/in.h"
#include"netdb.h"
#define PORT 4444
#define BUF_SIZE 2000
int main(int argc, char**argv) {
 struct sockaddr_in addr, cl_addr;
 int sockfd, ret;
 char buffer[BUF_SIZE];
 struct hostent * server;
 char * serverAddr;
 if (argc < 2) {
  printf("usage: client < ip address >\n");
  exit(1);        }
 serverAddr = argv[1];
 sockfd = socket(AF_INET, SOCK_STREAM, 0);
 if (sockfd < 0) {
  printf("Error creating socket!\n");
  exit(1);           }
 printf("Socket created...\n");
 memset(&addr, 0, sizeof(addr));
 addr.sin_family = AF_INET;
 addr.sin_addr.s_addr = inet_addr(serverAddr);
 addr.sin_port = PORT;
 ret = connect(sockfd, (struct sockaddr *) &addr, sizeof(addr));
 if (ret < 0) {
  printf("Error connecting to the server!\n");
  exit(1);     }
 printf("Connected to the server...\n");
 memset(buffer, 0, BUF_SIZE);
 printf("Enter your message(s): ");
 while (fgets(buffer, BUF_SIZE, stdin) != NULL) {
  ret = send(sockfd, buffer, BUF_SIZE, 0);
  if (ret < 0) {
   printf("Error sending data!\n\t-%s", buffer);
  }
  ret = recv(sockfd, buffer, BUF_SIZE, 0);
  if (ret < 0) {
   printf("Error receiving data!\n");
  }
     else              {
   printf("Received: ");
   fputs(buffer, stdout);
   printf("\n");    }
 }
 return 0;    }
```

OUTPUT

Running multiuserchat server

```
user@user-desktop:~/Desktop/multiuser$ gcc tcpservernew.c -o s
user@user-desktop:~/Desktop/multiuser$ ./s
Socket created...
Binding done...
Waiting for a connection...
Connection accepted...
Received data from 192.168.200.33: lmcst

Sent data to 192.168.200.33: lmcst

Connection accepted...
Received data from 192.168.200.33: s6cs students

Sent data to 192.168.200.33: s6cs students
```

Running multiuser chat client 1

```
user@user-desktop:~/Desktop/multiuser$ gcc tcpclientnew.c -o c1
user@user-desktop:~/Desktop/multiuser$ ./c1 192.168.200.33
Socket created...
Connected to the server...
Enter your message(s): lmcst
Received: lmcst
```

Running multiuser chat client 2

```
user@user-desktop:~/Desktop/multiuser$ gcc tcpclientnew.c -o c2
user@user-desktop:~/Desktop/multiuser$ ./c2 192.168.200.33
Socket created...
Connected to the server...
Enter your message(s): s6cs students
Received: s6cs students
```