**Problem definition :-**

Design and Develop a software product using socket programming concepts that allow the user to

perform group TEXT chatting. Include the following features.

a) Allows different user to register as a new member of a chat team.

b) Allows user to login and logout the chat room.

c) Allow the user to connect and disconnect the chat group.

d) Allows text chatting among all connected registered users.

e) There should be separate marking for text sent and received by different registered connected users.

f) Maintain the chat history for ‘n’ days, where ‘n’ is customizable.

g) Allow administrator to view chat history.

Note: Assume single group for chatting.

**Problem Requirements:-**

* Socket creation.
* Bind this socket with one IP address and one port.
* Start listening to that socket.
* Registration
* Deregistration
* Login.
* Client sends data.
* Server sends to all clients.
* Client displays the message.
* Client disconnects.

**Sequence used for Sending Message:-**

* As client gets connected to the server it transfers the order of matrix so that server and clients can share message synchronously.
* As order is received by server it goes in a loop and continuously waits for reading from the socket. At the same time client goes in a loop in order to send the message.
* As data is received by the server it sends message to other client.

**Error Encountered at the Time of Execution:-**

* Server can’t bind local address.-wait for two minute so that previous execution can leave that local address
* Data are not written properly- put printf statement after every line to find the error and accordingly take the action.
* Server is not reading properly – put null character after received data the display it.

**System Calls Used:-**

1. **Setsockopt :-**

*int setsockopt(int socket, int level, int option\_name,*

*const void \*option\_value, socklen\_t option\_len);*

The *setsockopt*() function shall set the option specified by the *option\_name* argument, at the protocol level specified by the *level* argument, to the value pointed to by the *option\_value*argument for the socket associated with the file descriptor specified by the *socket* argument.

The [*<sys/socket.h>*](http://pubs.opengroup.org/onlinepubs/009695399/basedefs/sys/socket.h.html) header defines the socket-level options. The options are as follows:

* SO\_DEBUG
* SO\_BROADCAST
* SO\_REUSEADDR

1. **Select :-**

***int select(int*** *nfds****, fd\_set \*****readfds****, fd\_set \*****writefds****,***

***fd\_set \*****exceptfds****, struct timeval \*****timeout****);***

#include <sys/select.h>

select() and pselect() allow a program to monitor multiple file

descriptors, waiting until one or more of the file descriptors become

"ready" for some class of I/O operation (e.g., input possible). A

file descriptor is considered ready if it is possible to perform a

corresponding I/O operation (e.g., read(2) without blocking, or a

sufficiently small write(2)).

1. **FD\_ISSET :-**

*int FD\_ISSET(int fd, fd\_set\* fdset)*

* + If you want to monitor more than 256 file descriptors on a single tpf\_select\_bsd call, the application must define its own FD\_SETSIZE before the include of header sys/time.h.
  + The purpose of the FD functions is to set up the input for and check the output from a tpf\_select\_bsd call. The FD\_CLR, FD\_COPY, FD\_SET, and FD\_ZERO functions set up the input for the tpf\_select\_bsd call. The FD\_ISSET function checks the output from the tpf\_select\_bsd call.

1. **Socket :-**

sockfd = socket(AF\_INET, SOCK\_STREAM, 0

#include <sys/socket.h>

*<sys/socket.h>* makes available a type, **socklen\_t**, which is an unsigned opaque integral type of length of at least 32 bits. To forestall portability problems, it is recommended that applications should not use values larger than 232-1.

SOCK\_DGRAM Datagram socket

SOCK\_STREAM Byte-stream socket

SOCK\_SEQPACKET Sequenced-packet socket

**Structure Used by System Calls :-**

**Sockaddr \_in**

**struct sockaddr\_in, struct in\_addr**

Structures for handling internet addresses

**Prototypes**

|  |
| --- |
| #include <netinet/in.h>  struct sockaddr\_in {  short sin\_family; // e.g. AF\_INET  unsigned short sin\_port; // e.g. htons(3490)  struct in\_addr sin\_addr; // see struct in\_addr, below  char sin\_zero[8]; // zero this if you want to  };  struct in\_addr {  unsigned long s\_addr; // load with inet\_aton()  }; |

**The timeout**

The time structures involved are defined in *<sys/time.h>*and look like

**Prototypes**

struct timeval {

long tv\_sec; /\* seconds \*/

long tv\_usec; /\* microseconds \*/

};

and

struct timespec {

long tv\_sec; /\* seconds \*/

long tv\_nsec; /\* nanoseconds \*/

};

**Creation of Socket**

* Create one socket and assign file descriptor to that socket.

if ((\*sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1)

{

Create Socket

perror("Socket"); exit(1);

}

* Assign address.

my\_addr->sin\_family = AF\_INET;

my\_addr->sin\_port = htons(5780);

my\_addr->sin\_addr.s\_addr = INADDR\_ANY;

memset(my\_addr->sin\_zero, '\0', sizeof my\_addr->sin\_zero);

* Define property to socket

if (setsockopt(\*sockfd, SOL\_SOCKET, SO\_REUSEADDR, &yes, sizeof(int)) == -1) { perror("setsockopt"); exit(1); }

Created Socket

Property

* **Bind address and socket together.**

if (bind(\*sockfd, (struct sockaddr \*)my\_addr, sizeof(struct sockaddr)) == -1) { perror("Unable to bind"); exit(1);

* **Listen to this address or socket for new client.**
  + - if (listen(\*sockfd, 10) == -1) { perror("listen"); exit(1); } printf("\nTCPServer Waiting for client on port 5780\n");
* **Now TCP is responsible for listening to new clients**

**Socket and TCP Overview :-**

Property

Create Socket

**Opening of File Descriptors**

Whenever TCP accepts new client connection it request server to create new file descriptor and server maintain the set of all file descriptor.

FD\_SET(sockfd, &master);

fdmax = sockfd;

while(1){

read\_fds = master;

if(select(fdmax+1, &read\_fds, NULL, NULL, NULL) == -1){

perror("select");

exit(4);

}

for (i = 0; i <= fdmax; i++){

if (FD\_ISSET(i, &read\_fds)){

if (i == sockfd){

connection\_accept(&master, &fdmax, sockfd, &client\_addr); }

else

send\_recv(i, &master, sockfd, fdmax); } } } return 0; }

**Accepting connection**

socklen\_t addrlen;

int newsockfd; addrlen = sizeof(struct sockaddr\_in);

if((newsockfd = accept(sockfd, (struct sockaddr \*)client\_addr, &addrlen)) == -1)

{ perror("accept"); exit(1); }

else { FD\_SET(newsockfd, master);

if(newsockfd > \*fdmax){

\*fdmax = newsockfd; }

printf("new connection from %s on port %d \n",inet\_ntoa(client\_addr->sin\_addr), ntohs(client\_addr->sin\_port));

}

**Socket Chatting Model**

**Server**

Create Socket

Property

Client1:-hello

Client1:-hello

Client1:-hello

Client1:-hello

hello

**Implementation of chat program**

**Program Features :-**

* **User registration.**
* **User deregistration.**
* **User Long in**
* **Exit**

**User Registration :-**

User want to register himself/herself for group cht

Checking criteria :- user name should not exist.

If user name does not exist ask user for a password.

Insert it into database.

Client server

sends new username

Search username

no, send 0

send password

Update database

User Deregistration

It removes the entry from the database.

Client server

Client server

sends new username

Search username

no, send 0

send password

Update database

**Server Side Programming :-**

#include <stdio.h>

#include<limits.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <netdb.h>

#define PORT 4950

#define BUFSIZE 1024

struct users{

int id;

char name[10];

int status;

} currentusers[10];

static int counter=0;

void printall(){

int k;

for(k=0;k<counter;k++){

printf("\nName %d:%s\nStatus %d:%d\nId %d:%d\n",k,currentusers[k].name,k,currentusers[k].status,k,currentusers[k].id);

}

}

int check\_user(char acname[BUFSIZE]){

FILE \*fp;

size\_t len = 0;

ssize\_t read;

char line[BUFSIZE];

fp=fopen("user\_data.txt","r");

while (( fread ( line,sizeof(line), 1, fp ))) {

if(strncmp(line,acname,(read-1))==0){

fclose(fp);

return 1;

}printf("Am in");

}

fclose(fp);

return 0;

}

validate\_user(int sockfd)

{

char recv\_buf[BUFSIZE],acname[BUFSIZE];

if ((recv(sockfd, recv\_buf, BUFSIZE, 0)) <= 0) {

printf("error\n");

}

//printf("%s\n",recv\_buf);

if(check\_user(recv\_buf)){

//He is registered so let him proceed

printf("Welcome %s",recv\_buf);

strcpy(currentusers[counter].name,recv\_buf);

if (send(sockfd,"1", sizeof("1"), 0) == -1) {

printf("error");

}

}

else{

//Now we know that he is not registered!

printf("%s You are not registered!",recv\_buf);//Just stopping him for time being

fflush(stdout);

strcpy(acname,recv\_buf);

if (send(sockfd,"0", sizeof("0"), 0) == -1) {

printf("error");

}

strncpy(recv\_buf,"\0",BUFSIZE);

if ((recv(sockfd, recv\_buf, BUFSIZE, 0)) <= 0) {

printf("error\n");

}

else{

FILE \*fp;

fp=fopen("user\_data.txt","a");

fwrite(acname,1,sizeof(acname),fp);

//fwrite("\n",1,sizeof("\n"),fp);

fclose(fp);

printf("Registered Successfully!\n");

strcpy(currentusers[counter].name,acname);

}

}

strncpy(recv\_buf,"\0",BUFSIZE);

fflush(stdout);

//printf("acname=%s\n",acname);

currentusers[counter].status=1;

currentusers[counter].id=sockfd;

//He is online

printall();

printf("\nName %d:%s\nStatus %d:%d\nId %d:%d\n",counter,currentusers[counter].name,counter,currentusers[counter].status,counter,currentusers[counter].id);

counter++;

}

void send\_to\_all(int j, int i, int sockfd, int nbytes\_recvd, char \*recv\_buf, fd\_set \*master,int cmd)

{

if (FD\_ISSET(j, master)){

if(cmd==1){

if(i==j){

if (send(j, recv\_buf, nbytes\_recvd, 0) == -1) {

perror("send");

}

}

}

else{

if (j != sockfd && j!=i) {

if (send(j, recv\_buf, nbytes\_recvd, 0) == -1) {

perror("send");

}

}

}

}

}

void send\_recv(int i, fd\_set \*master, int sockfd, int fdmax)

{

int nbytes\_recvd, j,k;

char list[100];

char recv\_buf[BUFSIZE], buf[BUFSIZE];

char userleft[100];

if ((nbytes\_recvd = recv(i, recv\_buf, BUFSIZE, 0)) <= 0) {

if (nbytes\_recvd == 0) {

printf("socket %d hung up\n", i);

for(k=0;k<=counter;k++){

if(i==currentusers[k].id && currentusers[k].status==1){

currentusers[k].status=0;

strcpy(userleft,currentusers[k].name);

strcat(userleft," has just left group chat!");

printf("\n%s just left group chat!\nStatus:%d\n",currentusers[k].name,currentusers[k].status);

for(j = 4; j <= fdmax; j++){

send\_to\_all(j, i, sockfd, sizeof(userleft), userleft, master,0);

}

}

}

}else {

perror("recv");

}

close(i);

FD\_CLR(i, master);

}else {

// printf("%s\n", recv\_buf);

//printf("server received %s.",recv\_buf);

if(strncmp(recv\_buf,"$cu",(nbytes\_recvd-1))==0){

printf("Check user began!");

//fflush(stdout);

printf("Sending list\n");

for(k=0;k<counter;k++){

if(currentusers[k].status==1)

sprintf(list,"%s",currentusers[k].name);

for(j =4; j<=fdmax; j++){

//printf("Sending it to id:%d",currentusers[k].id);

send\_to\_all(j, i,sockfd, sizeof(list),list, master,1);

}

strcpy(list,"\0");

}

fflush(stdout);

printf("Completed Sending list\n");

}

else

for(j = 0; j <= fdmax; j++){

send\_to\_all(j, i, sockfd, nbytes\_recvd, recv\_buf, master,0);

}

}

}

void connection\_accept(fd\_set \*master, int \*fdmax, int sockfd, struct sockaddr\_in \*client\_addr)

{

socklen\_t addrlen;

int newsockfd;

addrlen = sizeof(struct sockaddr\_in);

if((newsockfd = accept(sockfd, (struct sockaddr \*)client\_addr, &addrlen)) == -1) {

perror("accept");

exit(1);

}else {

FD\_SET(newsockfd, master);

if(newsockfd > \*fdmax){

\*fdmax = newsockfd;

}

printf("new connection from %s on port %d \n",inet\_ntoa(client\_addr->sin\_addr), ntohs(client\_addr->sin\_port));

}

validate\_user(newsockfd);

}

void connect\_request(int \*sockfd, struct sockaddr\_in \*my\_addr)

{

int yes = 1;

if ((\*sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("Socket");

exit(1);

}

my\_addr->sin\_family = AF\_INET;

my\_addr->sin\_port = htons(4950);

my\_addr->sin\_addr.s\_addr = INADDR\_ANY;

memset(my\_addr->sin\_zero, '\0', sizeof my\_addr->sin\_zero);

if (setsockopt(\*sockfd, SOL\_SOCKET, SO\_REUSEADDR, &yes, sizeof(int)) == -1) {

perror("setsockopt");

exit(1);

}

if (bind(\*sockfd, (struct sockaddr \*)my\_addr, sizeof(struct sockaddr)) == -1) {

perror("Unable to bind");

exit(1);

}

if (listen(\*sockfd, 10) == -1) {

perror("listen");

exit(1);

}

printf("\nTCPServer Waiting for client on port 4950\n");

fflush(stdout);

}

int main()

{

fd\_set master;

fd\_set read\_fds;

int fdmax, i;

int sockfd= 0;

struct sockaddr\_in my\_addr, client\_addr;

FD\_ZERO(&master);

FD\_ZERO(&read\_fds);

connect\_request(&sockfd, &my\_addr);

FD\_SET(sockfd, &master);

fdmax = sockfd;

while(1){

read\_fds = master;

if(select(fdmax+1, &read\_fds, NULL, NULL, NULL) == -1){

perror("select");

exit(4);

}

for (i = 0; i <= fdmax; i++){

if (FD\_ISSET(i, &read\_fds)){

if (i == sockfd){

connection\_accept(&master, &fdmax, sockfd, &client\_addr);

}

else

send\_recv(i, &master, sockfd, fdmax);

}

}

}

return 0;

}

**client Side Programming:-**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <errno.h>

#define BUFSIZE 1024

static flag=0;

char acname[BUFSIZE];

void send\_user(int sockfd)

{

char recv\_buf[BUFSIZE];

int response,ichoice;

//printf("sending name");

send(sockfd, acname, strlen(acname), 0);

recv(sockfd, recv\_buf, sizeof(recv\_buf), 0);

response=atoi(recv\_buf);

if(response==0){

printf("Do you wanna Signup?\n1:Yes\n2:No\n");

scanf("%d",&ichoice);

if(ichoice==1){

send(sockfd, acname, strlen(acname), 0);

response=1;

}

else{

printf("Bye!");

exit(0);

}

}

if(response==1)

printf("Welcome %s",acname);

fflush(stdout);

}

void send\_recv(int i, int sockfd)

{ flag++;

char send\_buf[BUFSIZE];

char recv\_buf[BUFSIZE];

int nbyte\_recvd;

char acname1[BUFSIZE];

if (i == 0){

//printf("\nU:");

fgets(send\_buf, BUFSIZE, stdin);

if (strcmp(send\_buf , "quit\n") == 0) {

exit(0);

}else if (strcmp(send\_buf , "$cu\n") == 0) {

//printf("client sending %s\n","$cu");

send(sockfd,"$cu\n", strlen("$cu\n"), 0);

//fflush(stdout);

//printf("Completed sending");

//fflush(stdout);

}else{

strcpy(acname1,acname);

strcat(acname1,":");

strcat(acname1,send\_buf);

strcpy(send\_buf,acname1);

send(sockfd, send\_buf, strlen(send\_buf), 0);

strcpy(acname1,"\0");

}

}else {

nbyte\_recvd = recv(sockfd, recv\_buf, BUFSIZE, 0);

recv\_buf[nbyte\_recvd] = '\0';

printf("%s\n",recv\_buf);

fflush(stdout);

}

}

void connect\_request(int \*sockfd, struct sockaddr\_in \*server\_addr)

{

if ((\*sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("Socket");

exit(1);

}

server\_addr->sin\_family = AF\_INET;

server\_addr->sin\_port = htons(4950);

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

memset(server\_addr->sin\_zero, '\0', sizeof server\_addr->sin\_zero);

if(connect(\*sockfd, (struct sockaddr \*)server\_addr, sizeof(struct sockaddr)) == -1) {

perror("connect");

exit(1);

}

}

int main()

{

int sockfd, fdmax, i;

struct sockaddr\_in server\_addr;

fd\_set master;

fd\_set read\_fds;

connect\_request(&sockfd, &server\_addr);

FD\_ZERO(&master);

FD\_ZERO(&read\_fds);

FD\_SET(0, &master);

FD\_SET(sockfd, &master);

fdmax = sockfd;

printf("\nName:");

scanf("%s",acname);

getchar();

send\_user(sockfd);

while(1){

read\_fds = master;

if(select(fdmax+1, &read\_fds, NULL, NULL, NULL) == -1){

perror("select");

exit(4);

}

for(i=0; i <= fdmax; i++ )

if(FD\_ISSET(i, &read\_fds)){

//printf("%s:",acname);

send\_recv(i, sockfd);

}

}

printf("client-quited\n");

close(sockfd);

return 0;

}

**Sample Input Output**

Case 1: Registration

MENU

1 Registration

2 Deregistration

3 Login

4 exit

1

Enter user name.

Abc

Please enter password.

qwer

Registered successful.

Case2: Deregistration

MENU

1 Registration

2 Deregistration

3 Login

4 exit

2

Enter user name to be deregidtered.

Abc

Enter password.

Qwer

Deregister Successful

**References**

* **Unix System Programming using C++**
* **U.P.K sir’s code**

**Thank YOU…**