**Assignment - 5**

5. Write a program in C using thread library and TCP sockets to build a chat server which enable clients communicating to each other through the chat server. Message logs must be maintained in the server in a text file. Each client will see the conversations in real time. Clients must handled by a server thread.

**Solution:**

**Server.c Program**

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <pthread.h>

#include <sys/types.h>

#include <signal.h>

#define MAX\_CONNECTIONS 5

#define BUF\_SIZE 1024

#define NAME\_LEN 255

#define INVALID 0;

unsigned int ip\_to\_int (const char \* ip);

static \_Atomic unsigned int cli\_count = 0;

// structure for the clients

typedef struct client

{

struct sockaddr\_in address;

int sockfd;

int cid;

char name[NAME\_LEN];

} CLIENT;

CLIENT \*clients[MAX\_CONNECTIONS];

pthread\_mutex\_t clients\_mutex = PTHREAD\_MUTEX\_INITIALIZER;

// --------------------------------------------------------------------------------------------------------

long random\_at\_most(long max)

{

unsigned long

num\_bins = (unsigned long)max + 1,

num\_rand = (unsigned long)RAND\_MAX + 1,

bin\_size = num\_rand / num\_bins,

defect = num\_rand % num\_bins;

long x;

do

{

x = random();

} while (num\_rand - defect <= (unsigned long)x);

return x / bin\_size;

}

// -----------------------------------------------------------------------------------------------------------

void str\_trim(char \*arr, int len)

{

int i;

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

{

if (arr[i] == '\n')

{

arr[i] = '\0';

break;

}

}

}

void add\_client(CLIENT \*cli)

{

pthread\_mutex\_lock(&clients\_mutex);

int i;

for (i = 0; i < MAX\_CONNECTIONS; i++)

{

if (!clients[i])

{

clients[i] = cli;

break;

}

}

pthread\_mutex\_unlock(&clients\_mutex);

}

void rem\_client(int cid)

{

pthread\_mutex\_lock(&clients\_mutex);

int i;

for (i = 0; i < MAX\_CONNECTIONS; i++)

{

if (clients[i])

{

if (clients[i]->cid == cid)

{

clients[i] = NULL;

break;

}

}

}

pthread\_mutex\_unlock(&clients\_mutex);

}

void snd\_msg(char \*s, int cid)

{

pthread\_mutex\_lock(&clients\_mutex);

int i;

for (i = 0; i < MAX\_CONNECTIONS; i++)

{

if (clients[i])

{

if (clients[i]->cid != cid)

{

if (write(clients[i]->sockfd, s, strlen(s)) < 0)

{

printf("Error writing\n");

break;

}

}

}

}

pthread\_mutex\_unlock(&clients\_mutex);

}

void \*handleClient(void \*client)

{

char buff[BUF\_SIZE];

char name[NAME\_LEN];

int leave\_flag = 0;

cli\_count++;

CLIENT \*cli = (CLIENT \*)client;

recv(cli->sockfd, name, NAME\_LEN, 0);

strcpy(cli->name, name);

sprintf(buff, "%s has joined\n", cli->name);

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

snd\_msg(buff, cli->cid);

bzero(buff, BUF\_SIZE);

char id[3];

FILE \*fp;

char \*fileName = "Logs.txt";

while (1)

{

if (leave\_flag == 1)

{

break;

}

int r = recv(cli->sockfd, buff, BUF\_SIZE, 0);

if (r > 0)

{

if (strlen(buff) > 0)

{

snd\_msg(buff, cli->cid);

str\_trim(buff, strlen(buff));

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

//----------------------------------------------

sprintf(id, "%d", cli->cid);

fp = fopen(fileName, "a+");

fputs(id, fp);

fputc(' ', fp);

fputs(buff, fp);

fputc('\n', fp);

fclose(fp);

//------------------------------------------------

}

}

else if (r == 0 || strcmp(buff, "exit") == 0)

{

sprintf(buff, "%s has left", cli->name);

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

snd\_msg(buff, cli->cid);

leave\_flag = 1;

}

else

{

printf("Error handle client\n");

leave\_flag = 1;

}

bzero(buff, BUF\_SIZE);

}

close(cli->sockfd);

rem\_client(cli->cid);

free(cli);

pthread\_detach(pthread\_self());

}

int main(int argc, char \*argv[])

{

if (argc < 2)

{

printf("Port no found.\n");

return -1;

}

// setting the port and neccessary variables

int port = atoi(argv[1]);

int ser\_sock\_fd, new\_sock\_fd;

pthread\_t tid;

int option = 1;

struct sockaddr\_in ser\_add, cli\_add;

socklen\_t cli\_add\_size;

// socket settings

ser\_sock\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

ser\_add.sin\_family = AF\_INET;

ser\_add.sin\_addr.s\_addr = inet\_addr("10.0.0.1");

ser\_add.sin\_port = htons(port);

// signals

signal(SIGPIPE, SIG\_IGN);

if (setsockopt(ser\_sock\_fd, SOL\_SOCKET, (SO\_REUSEPORT | SO\_REUSEADDR), (char \*)&option, sizeof(option)) < 0)

{

printf("Error setsockopt\n");

return -1;

}

// binding socket

if (bind(ser\_sock\_fd, (struct sockaddr \*)&ser\_add, sizeof(ser\_add)) < 0)

{

printf("Bind Error\n");

return -1;

}

// listening

if (listen(ser\_sock\_fd, MAX\_CONNECTIONS) < 0)

{

printf("Listening error\n");

return -1;

}

printf("--------------Chat Server Started-------------\n");

while (1)

{

cli\_add\_size = sizeof(cli\_add);

new\_sock\_fd = accept(ser\_sock\_fd, (struct sockaddr \*)&cli\_add, &cli\_add\_size);

struct sockaddr\_in \*cliIP = (struct sockaddr\_in \*)&cli\_add;

struct in\_addr ipAddr = cliIP->sin\_addr;

char str[INET\_ADDRSTRLEN];

inet\_ntop(AF\_INET, &ipAddr, str, INET\_ADDRSTRLEN);

CLIENT \*newCli = (CLIENT \*)malloc(sizeof(CLIENT));

newCli->address = cli\_add;

newCli->sockfd = new\_sock\_fd;

//

// newCli->cid = random\_at\_most(100);

newCli->cid = ip\_to\_int(str);

add\_client(newCli);

pthread\_create(&tid, NULL, &handleClient, (void \*)newCli);

sleep(1);

}

return 0;

}

unsigned int ip\_to\_int (const char \* ip)

{

/\* The return value. \*/

unsigned v = 0;

/\* The count of the number of bytes processed. \*/

int i;

/\* A pointer to the next digit to process. \*/

const char \* start;

start = ip;

for (i = 0; i < 4; i++) {

/\* The digit being processed. \*/

char c;

/\* The value of this byte. \*/

int n = 0;

while (1) {

c = \* start;

start++;

if (c >= '0' && c <= '9') {

n \*= 10;

n += c - '0';

}

/\* We insist on stopping at "." if we are still parsing

the first, second, or third numbers. If we have reached

the end of the numbers, we will allow any character. \*/

else if ((i < 3 && c == '.') || i == 3) {

break;

}

else {

return INVALID;

}

}

if (n >= 256) {

return INVALID;

}

v \*= 256;

v += n;

}

return v;

}

**Client.c - Program**

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <pthread.h>

#include <sys/types.h>

#include <signal.h>

#define MAX\_CONNECTIONS 5

#define BUF\_SIZE 1024

#define NAME\_LEN 255

#define MSG\_LEN 2048

volatile sig\_atomic\_t flag = 0;

int sockfd = 0;

char name[NAME\_LEN];

void str\_trim(char \*arr, int len)

{

int i;

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

{

if (arr[i] == '\n')

{

arr[i] = '\0';

break;

}

}

}

void \*rcvHandler()

{

char msg[MSG\_LEN] = {};

int rcv;

while (1)

{

rcv = recv(sockfd, msg, BUF\_SIZE, 0);

if (rcv > 0)

{

printf("%s", msg);

}

else if (rcv == 0)

{

break;

}

bzero(msg, BUF\_SIZE);

rcv = 0;

}

}

void \*sndHandler()

{

char buff[BUF\_SIZE] = {};

char msg[MSG\_LEN] = {};

while (1)

{

fgets(buff, BUF\_SIZE, stdin);

str\_trim(buff, BUF\_SIZE);

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

{

flag = 1;

break;

}

else

{

sprintf(msg, "%s : %s\n", name, buff);

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

}

bzero(buff, BUF\_SIZE);

bzero(msg, MSG\_LEN);

}

}

int main(int argc, char \*argv[])

{

if (argc < 2)

{

printf("Port no found.\n");

return -1;

}

int port = atoi(argv[1]);

printf("Your name: ");

fgets(name, NAME\_LEN, stdin);

str\_trim(name, strlen(name));

struct sockaddr\_in ser\_add;

socklen\_t cli\_add\_size;

// socket settings

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

ser\_add.sin\_family = AF\_INET;

ser\_add.sin\_addr.s\_addr = inet\_addr("10.0.0.1");

ser\_add.sin\_port = htons(port);

// connect to server

if (connect(sockfd, (struct sockaddr \*)&ser\_add, sizeof(ser\_add)) < 0)

{

printf("connection error\n");

return 0;

}

printf("-------------Connected to the server-----------\n");

send(sockfd, name, NAME\_LEN, 0);

printf("--------------Chat Server Started-------------\n");

pthread\_t sendThread;

pthread\_t recvThread;

if (pthread\_create(&sendThread, NULL, &sndHandler, NULL) != 0)

{

printf("Error send thread\n");

return -1;

}

if (pthread\_create(&sendThread, NULL, &rcvHandler, NULL) != 0)

{

printf("Error recv thread\n");

return -1;

}

while (1)

{

if (flag)

{

printf("\nBye\n");

break;

}

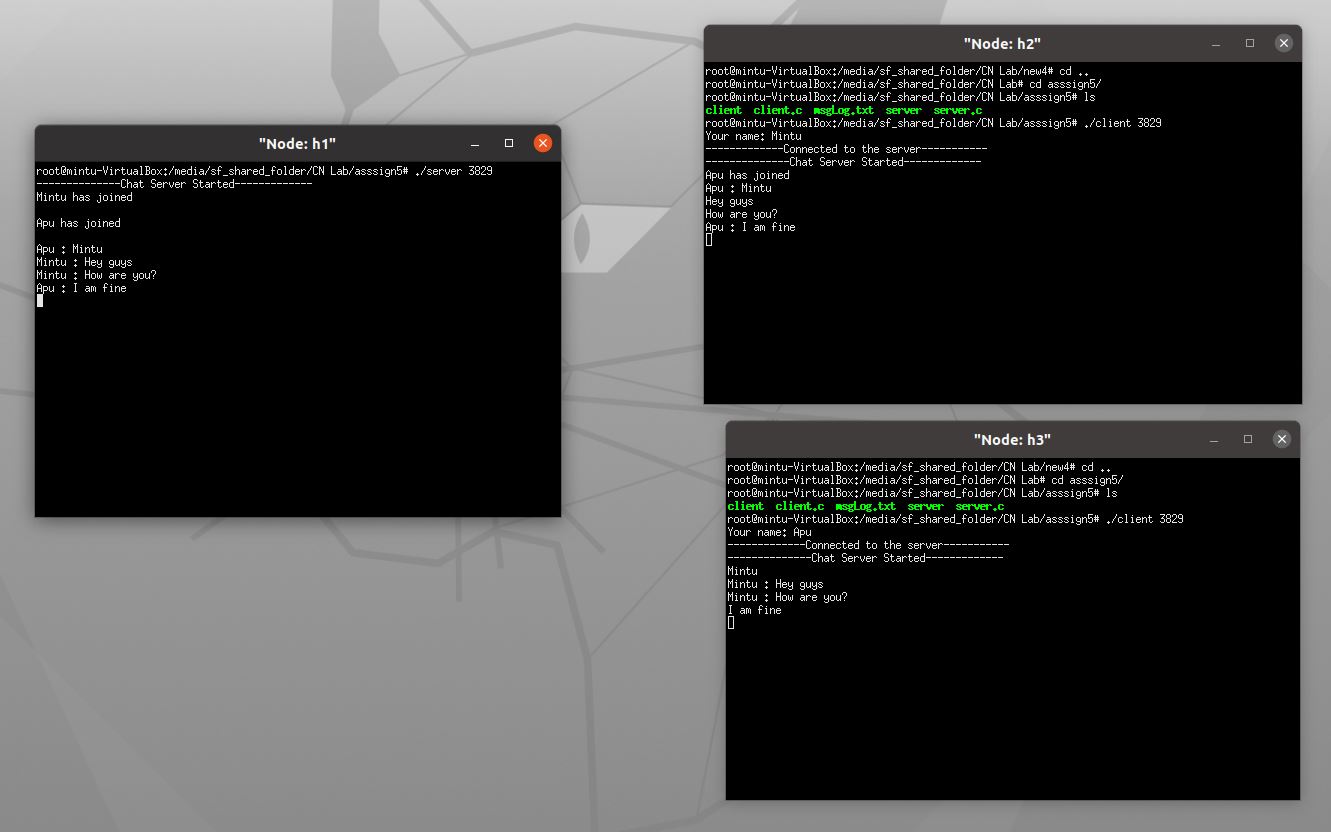
}

close(sockfd);

return 0;

}

**Output:**

****