## 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: