ASSIGNMENT-7

Name: Sayantan Roy

Roll Number: GCECTB-R18-3025

Subject: Computer Network Lab

Dept: CSE

Year:3rd

ASSIGNMENT-7

TCP MULTICLIENT CHAT SERVER IMPLEMENTATION IN C

System used :kali linux 2021.2
Kernel:4.19.152
Desktop Environment: Gnome 3.38

ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to my teacher **Mr. Pinaki Mukherjee and Mrs. Sucharita Mondal**who gave me the golden opportunity to do this wonderful project on the topic which also helped me in doing a lot of Research and i came to know about so many new things I am really thankful to them.

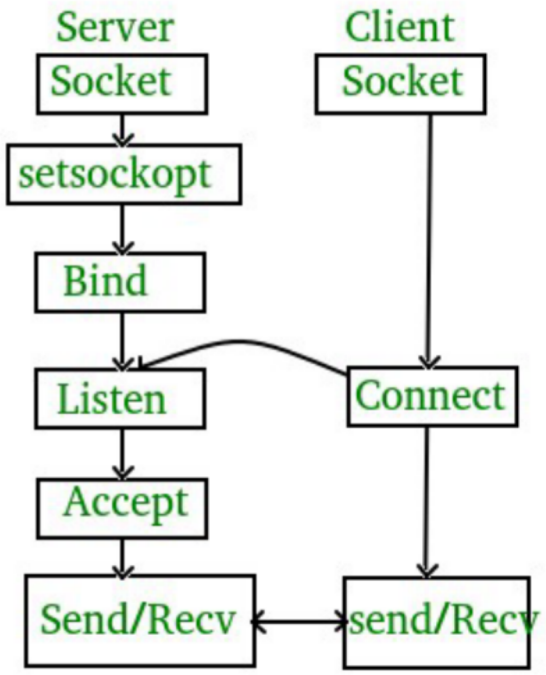
Secondly i would also like to thank my all the teachers parents and friends who helped me a lot in finalizing this project within the limited time frame.

-Sayantan Roy
GCECTB-R18-3025

TCP MULTICLIENT CHAT IMPLEMENTATION IN C

If we are creating a connection between client and server using TCP then it has few functionality like, TCP is suited for applications that require high reliability, and transmission time is relatively less critical. It is used by other protocols like HTTP, HTTPs, FTP, SMTP, Telnet. TCP rearranges data packets in the order specified. There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent. TCP does Flow Control and requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control. It also does error checking and error recovery. Erroneous packets are retransmitted from the source to the destination.

The entire process can be broken down into following steps:



The entire process can be broken down into following steps:

TCP Server –

1. using create(), Create TCP socket.
2. using bind(), Bind the socket to server address.
3. using listen(), put the server socket in a passive mode, where it waits for the client to approach the server to make a connection
4. using accept(), At this point, connection is established between client and server, and they are ready to transfer data.
5. Go back to Step 3.

TCP Client –

1. Create TCP socket.
2. connect newly created client socket to server.

SERVERSIDE PPROGRAM:

FILENAME:tcpmultichatserver.c

```
#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_CLIENTS 100
#define BUFFER_SZ 2048

static _Atomic unsigned int cli_count = 0;
static int uid = 10;

/* Client structure */
typedef struct{
        struct sockaddr_in address;
        int sockfd;
        int uid;
        char name[32];
} client_t;

client_t *clients[MAX_CLIENTS];

pthread_mutex_t clients_mutex = PTHREAD_MUTEX_INITIALIZER;
```

```
void str_overwrite_stdout() {
    printf("\r%s", "> ");
    fflush(stdout);
}

void str_trim_lf (char* arr, int length) {
  int i;
  for (i = 0; i < length; i++) { // trim \n
    if (arr[i] == '\n') {
      arr[i] = '\0';
      break;
    }
  }
}

void print_client_addr(struct sockaddr_in addr){
    printf("%d.%d.%d.%d",
        addr.sin_addr.s_addr & 0xff,
        (addr.sin_addr.s_addr & 0xff00) >> 8,
        (addr.sin_addr.s_addr & 0xff0000) >> 16,
        (addr.sin_addr.s_addr & 0xff000000) >> 24);
}

/* Add clients to queue */
void queue_add(client_t *cl){
            pthread_mutex_lock(&clients_mutex);

            for(int i=0; i < MAX_CLIENTS; ++i){
                        if(!clients[i]){
                                    clients[i] = cl;
                                    break;
                        }
```

```
	}

	pthread_mutex_unlock(&clients_mutex);
}

/* Remove clients to queue */
void queue_remove(int uid){
	pthread_mutex_lock(&clients_mutex);

	for(int i=0; i < MAX_CLIENTS; ++i){
		if(clients[i]){
			if(clients[i]->uid == uid){
				clients[i] = NULL;
				break;
			}
		}
	}

	pthread_mutex_unlock(&clients_mutex);
}

/* Send message to all clients except sender */
void send_message(char *s, int uid){
	pthread_mutex_lock(&clients_mutex);

	for(int i=0; i<MAX_CLIENTS; ++i){
		if(clients[i]){
			if(clients[i]->uid != uid){
				if(write(clients[i]->sockfd, s, strlen(s)) < 0){
					perror("ERROR: write to descriptor failed");
					break;
				}
```

```
                        }
                }
        }

        pthread_mutex_unlock(&clients_mutex);
}

/* Handle all communication with the client */
void *handle_client(void *arg){
        char buff_out[BUFFER_SZ];
        char name[32];
        int leave_flag = 0;

        cli_count++;
        client_t *cli = (client_t *)arg;

        // Name
        if(recv(cli->sockfd, name, 32, 0) <= 0 || strlen(name) <  2 || strlen(name) >= 32-1){
                printf("Didn't enter the name.\n");
                leave_flag = 1;
        } else{
                strcpy(cli->name, name);
                sprintf(buff_out, "%s has joined\n", cli->name);
                printf("%s", buff_out);
                send_message(buff_out, cli->uid);
        }

        bzero(buff_out, BUFFER_SZ);

        while(1){
                if (leave_flag) {
                        break;
```

```
            }

            int receive = recv(cli->sockfd, buff_out, BUFFER_SZ, 0);
            if (receive > 0){
                if(strlen(buff_out) > 0){
                    send_message(buff_out, cli->uid);

                    str_trim_lf(buff_out, strlen(buff_out));
                    printf("%s -> %s\n", buff_out, cli->name);
                }
            } else if (receive == 0 || strcmp(buff_out, "exit") == 0){
                sprintf(buff_out, "%s has left\n", cli->name);
                printf("%s", buff_out);
                send_message(buff_out, cli->uid);
                leave_flag = 1;
            } else {
                printf("ERROR: -1\n");
                leave_flag = 1;
            }

            bzero(buff_out, BUFFER_SZ);
        }

  /* Delete client from queue and yield thread */
        close(cli->sockfd);
  queue_remove(cli->uid);
  free(cli);
  cli_count--;
  pthread_detach(pthread_self());

        return NULL;
}
```

```
int main(int argc, char **argv){
        if(argc != 2){
                printf("Usage: %s <port>\n", argv[0]);
                return EXIT_FAILURE;
        }

        char *ip = "127.0.0.1";
        int port = atoi(argv[1]);
        int option = 1;
        int listenfd = 0, connfd = 0;
  struct sockaddr_in serv_addr;
  struct sockaddr_in cli_addr;
  pthread_t tid;

  /* Socket settings */
  listenfd = socket(AF_INET, SOCK_STREAM, 0);
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = inet_addr(ip);
  serv_addr.sin_port = htons(port);

  /* Ignore pipe signals */
        signal(SIGPIPE, SIG_IGN);

        if(setsockopt(listenfd, SOL_SOCKET,(SO_REUSEPORT | SO_REUSEADDR),
(char*)&option,sizeof(option)) < 0){
                perror("ERROR: setsockopt failed");
    return EXIT_FAILURE;
        }

        /* Bind */
  if(bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) < 0) {
```

```
  perror("ERROR: Socket binding failed");
  return EXIT_FAILURE;
}

/* Listen */
if (listen(listenfd, 10) < 0) {
  perror("ERROR: Socket listening failed");
  return EXIT_FAILURE;
        }

        printf("=== WELCOME TO THE CHATROOM ===\n");

        while(1){
                socklen_t clilen = sizeof(cli_addr);
                connfd = accept(listenfd, (struct sockaddr*)&cli_addr, &clilen);

                /* Check if max clients is reached */
                if((cli_count + 1) == MAX_CLIENTS){
                        printf("Max clients reached. Rejected: ");
                        print_client_addr(cli_addr);
                        printf(":%d\n", cli_addr.sin_port);
                        close(connfd);
                        continue;
                }

                /* Client settings */
                client_t *cli = (client_t *)malloc(sizeof(client_t));
                cli->address = cli_addr;
                cli->sockfd = connfd;
                cli->uid = uid++;

                /* Add client to the queue and fork thread */
```

```
            queue_add(cli);
            pthread_create(&tid, NULL, &handle_client, (void*)cli);

            /* Reduce CPU usage */
            sleep(1);
        }

        return EXIT_SUCCESS;
}
```

CLIENTSIDE PROGRAM:

FILENAME:tcpmultichatclient.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pthread.h>

#define LENGTH 2048

// Global variables
volatile sig_atomic_t flag = 0;
int sockfd = 0;
char name[32];

void str_overwrite_stdout() {
  printf("%s", "> ");
  fflush(stdout);
}

void str_trim_lf (char* arr, int length) {
  int i;
  for (i = 0; i < length; i++) { // trim \n
    if (arr[i] == '\n') {
      arr[i] = '\0';
      break;
    }
```

```
  }
}

void catch_ctrl_c_and_exit(int sig) {
    flag = 1;
}

void send_msg_handler() {
  char message[LENGTH] = {};
        char buffer[LENGTH + 32] = {};

  while(1) {
        str_overwrite_stdout();
    fgets(message, LENGTH, stdin);
    str_trim_lf(message, LENGTH);

    if (strcmp(message, "exit") == 0) {
                        break;
    } else {
      sprintf(buffer, "%s: %s\n", name, message);
      send(sockfd, buffer, strlen(buffer), 0);
    }

                bzero(message, LENGTH);
    bzero(buffer, LENGTH + 32);
  }
  catch_ctrl_c_and_exit(2);
}

void recv_msg_handler() {
        char message[LENGTH] = {};
  while (1) {
```

```
                int receive = recv(sockfd, message, LENGTH, 0);
    if (receive > 0) {
      printf("%s", message);
      str_overwrite_stdout();
    } else if (receive == 0) {
                        break;
    } else {
                        // -1
                }
                memset(message, 0, sizeof(message));
  }
}

int main(int argc, char **argv){
        if(argc != 2){
                printf("Usage: %s <port>\n", argv[0]);
                return EXIT_FAILURE;
        }

        char *ip = "127.0.0.1";
        int port = atoi(argv[1]);

        signal(SIGINT, catch_ctrl_c_and_exit);

        printf("Please enter your name: ");
  fgets(name, 32, stdin);
  str_trim_lf(name, strlen(name));


        if (strlen(name) > 32 || strlen(name) < 2){
                printf("Name must be less than 30 and more than 2 characters.\n");
                return EXIT_FAILURE;
```

```
        }

        struct sockaddr_in server_addr;

        /* Socket settings */
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = inet_addr(ip);
server_addr.sin_port = htons(port);


// Connect to Server
int err = connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr));
if (err == -1) {
                printf("ERROR: connect\n");
                return EXIT_FAILURE;
        }

        // Send name
        send(sockfd, name, 32, 0);

        printf("=== WELCOME TO THE CHATROOM ===\n");

        pthread_t send_msg_thread;
if(pthread_create(&send_msg_thread, NULL, (void *) send_msg_handler, NULL) != 0){
                printf("ERROR: pthread\n");
  return EXIT_FAILURE;
        }

        pthread_t recv_msg_thread;
if(pthread_create(&recv_msg_thread, NULL, (void *) recv_msg_handler, NULL) != 0){
                printf("ERROR: pthread\n");
```

```
                return EXIT_FAILURE;
        }

        while (1){
                if(flag){
                        printf("\nBye\n");
                        break;
   }
        }

        close(sockfd);

        return EXIT_SUCCESS;
}
```

COMPILATION:

```
┌──(tom㉿localhost)-[~]
└─$ gcc -Wall -g3 -fsanitize=address -pthread tcpmultichatserver.c -o tcpmultichatserver

┌──(tom㉿localhost)-[~]
└─$ gcc -Wall -g3 -fsanitize=address -pthread tcpmultichatclient.c -o tcpmultichatclient
```

OUTPUT:

output of server :

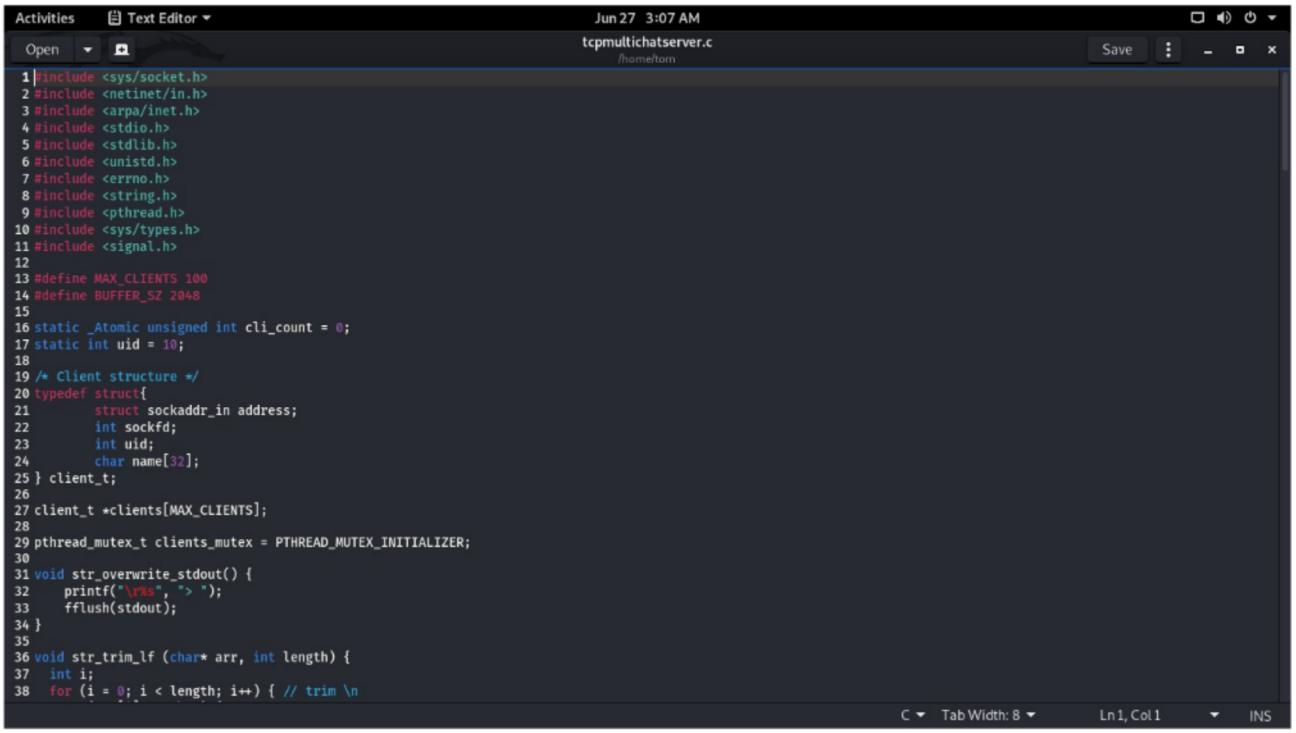
```
┌──(tom㉿localhost)-[~]
└─$ ./tcpmultichatserver 8888
=== WELCOME TO THE CHATROOM ===
sayantan has joined
tom has joined
tom: hi! -> tom
sayantan: hello ! how are you ! -> sayantan
```

output from client 1 (sayantan)

```
──(tom㉿localhost)-[~]
└─$ ./tcpmultichatclient 8888
Please enter your name: sayantan
=== WELCOME TO THE CHATROOM ===
> tom has joined
> tom: hi!
> hello ! how are you !
>
```

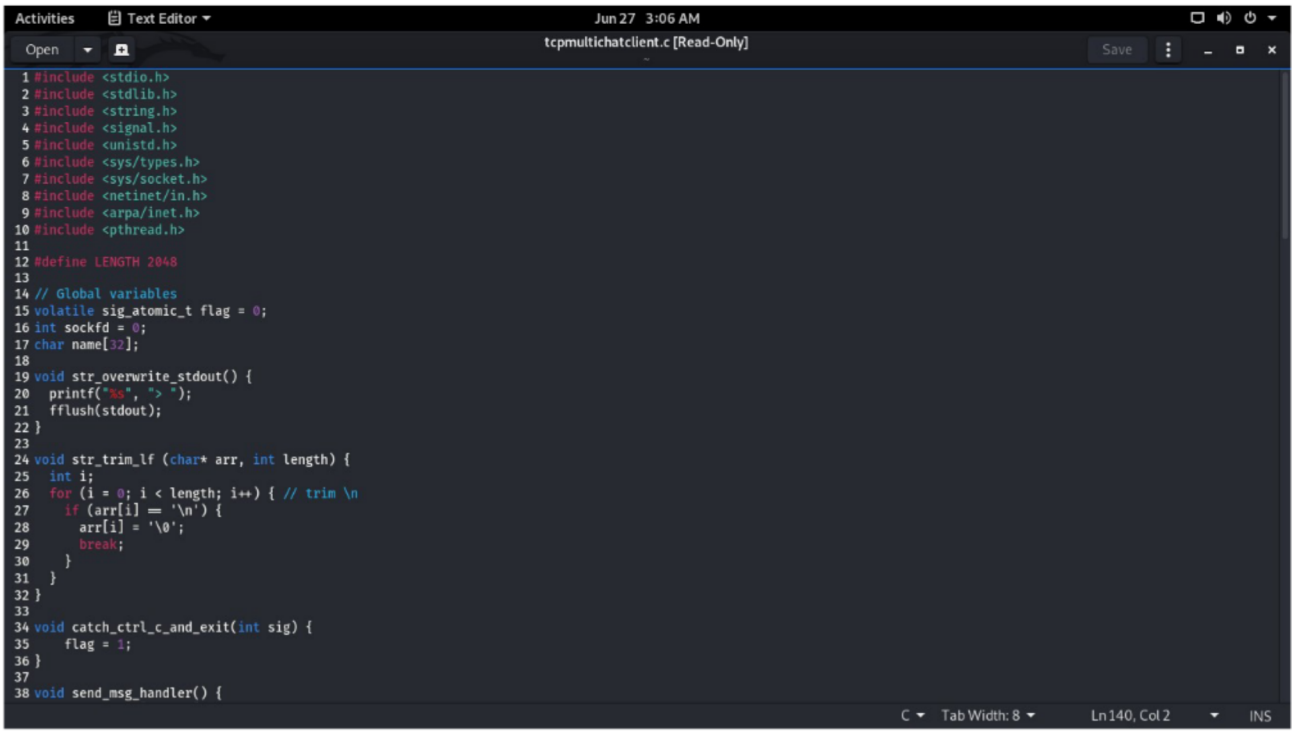
output from client 2 (tom)

```
──(tom㉿localhost)-[~]
└─$ ./tcpmultichatclient 8888
Please enter your name: tom
=== WELCOME TO THE CHATROOM ===
> hi!
> sayantan: hello ! how are you !
>
```




```c
#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_CLIENTS 100
#define BUFFER_SZ 2048

static _Atomic unsigned int cli_count = 0;
static int uid = 10;

/* Client structure */
typedef struct{
        struct sockaddr_in address;
        int sockfd;
        int uid;
        char name[32];
} client_t;

client_t *clients[MAX_CLIENTS];

pthread_mutex_t clients_mutex = PTHREAD_MUTEX_INITIALIZER;

void str_overwrite_stdout() {
    printf("\r%s", "> ");
    fflush(stdout);
}

void str_trim_lf (char* arr, int length) {
  int i;
  for (i = 0; i < length; i++) { // trim \n
```

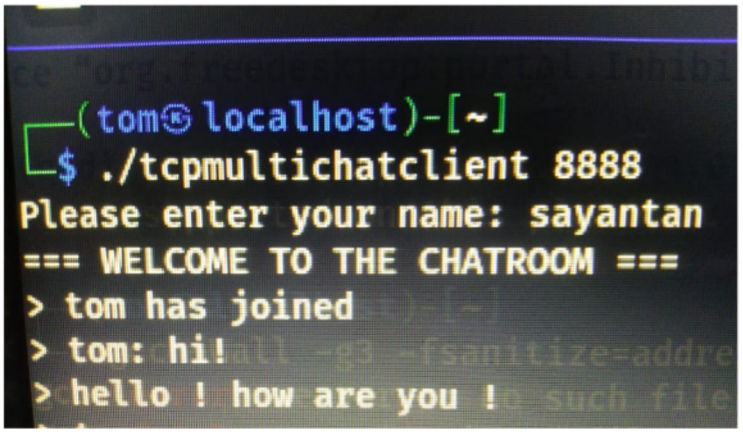
FILENAME:tcpmultichatserver.c



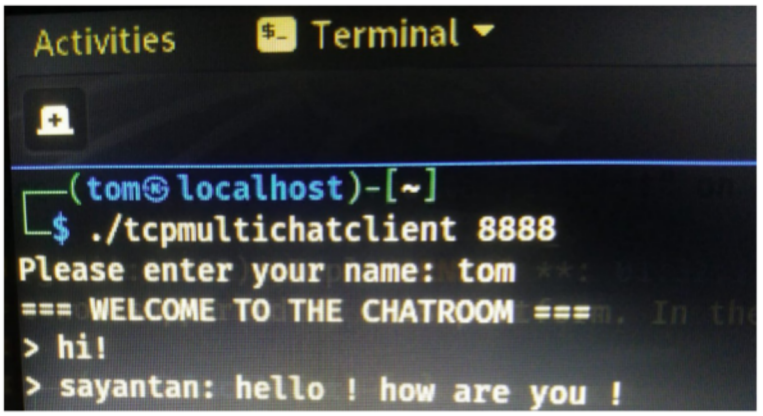

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pthread.h>

#define LENGTH 2048

// Global variables
volatile sig_atomic_t flag = 0;
int sockfd = 0;
char name[32];

void str_overwrite_stdout() {
  printf("%s", "> ");
  fflush(stdout);
}

void str_trim_lf (char* arr, int length) {
  int i;
  for (i = 0; i < length; i++) { // trim \n
    if (arr[i] == '\n') {
      arr[i] = '\0';
      break;
    }
  }
}

void catch_ctrl_c_and_exit(int sig) {
    flag = 1;
}

void send_msg_handler() {
```

FILENAME: tcpmultichatclient.c

TCP MULTICLIENT CHAT SERVER IS RUNNING AT PORT 8888



CLIENT 1 " sayantan" IS CONNECTED AT PORT 8888



CLIENT 2 "tom" IS CONNECTED AT PORT 8888

HERE CLIENT 1 AND CLIENT 2 HAVE JOINED AND STARTED CHATTING