UCS 1511 - Network Lab
Chat using TCP

Name:	Mahesh Bharadwaj K
Reg No:	185001089
Semester:	V
Date:	September 11, 2020

Aim:

To write a socket program to perform chat with multiple clients using TCP.

Algorithm

1. Server

1. Create a socket descriptor with **socket()** system call with AF_INET (IPV4 domain), SOCK_STREAM, default protocol and store as sockfd.
2. If sockfd is a negative number, socket creation failed, end program.
3. Create sockaddr_in object to assign IP address and Port number for socket. Set family to AF_INET, IP address to INADDR_ANY to accept connections from any client and port number required.
4. Bind newly created socket to addresss given in sockaddr_in.
5. If bind is non zero, bind failed, print error message and terminate.
6. Listen on the socked defined for as many clients as required. If **listen()** returns non zero value, print error message and terminate.
7. LOOP:
   (a) Clear readfds using **FD_ZERO()**
   (b) Add server socket descriptor to FD_SET readfds.
   (c) check for activity on socket using **select()** system call.
   (d) IF activity < 0, print error message and terminate
   (e) Check if sockfd (socket descriptor) is readfds set using **FD_ISSET()**;
   IF true,
   - New connection detected, **accept()** new client and get client descriptor
   - Store new client descriptor in array of client descriptors
   - Remove server socket descriptor from readfds.

   (f) Loop through all client descriptors and add them to the FD_SET readfds.
   (g) Check for activity using **select()** command.
   IF detected
   - Loop through clients one by one and check if they are in the set using **FD_ISSET()**.
   - Read message into buffer using **read()** command
   - IF message read is **'END'**, remove this client from client array
   - ELSE, display the message and send response to the client using **write()**.
8. Close connections on socket using **close()** and terminate program.

2. Client

1. Create a socket descriptor with **socket()** system call with AF_INET (IPV4 domain), SOCK_STREAM, default protocol and store as sockfd.
2. If sockfd is a negative number, socket creation failed, end program.
3. Create sockaddr_in object to assign IP address and Port number for socket. Set family to AF_INET, IP address to localhost(127.0.0.1) to connect to server and port number required.
4. Connect the client to server at address given in socket descriptor using **connect()** system call.
5. If connect() returns -1, connection failed; Print error message and terminate the program.
6. Read message from user into buffer variable.
7. IF the message is **END**, terminate the program and close the socket descriptor using **close()** system call.
8. ELSE, write message into server socket using **write()** system call.
9. Read the response from server into buffer variable using **read()** system call and display received message to the user.
10. Repeat from step 6.

Program

1. Server Side

```c
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <string.h>

#define MAX 256
#define CLIENT 5
#define SA struct sockaddr

int fd_max(const int arr[CLIENT], int sock_fd, fd_set *readfds)
{
    //Finding the largest file descriptor, required for select()

    int max = -1;
    for (int i = 0; i < CLIENT; ++i)
    {
        max = (arr[i] > max) ? arr[i] : max;

        // If it is valid, add it to set of descriptors
        if (arr[i] > 0)
            FD_SET(arr[i], readfds);
    }

    max = (sock_fd > max) ? sock_fd : max;

    return max + 1;
}

int main(int argc, char **argv)
```

```
{
    if(argc < 2){
        fprintf(stderr, "Please pass port number as second argument!\n");
        exit(EXIT_FAILURE);
    }

    int PORT = atoi(argv[1]);

    int sockfd, new_fd[CLIENT] = {0}, len;
    struct sockaddr_in servaddr, cli;
    char buff[MAX];

    // Setting timeouts for select()
    struct timeval tv;
    tv.tv_sec = 1;
    tv.tv_usec = 0;

    // Tracking activity on descriptors
    fd_set readfds;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1)
    {
        fprintf(stderr, "Socket creation failed!\n");
        exit(EXIT_FAILURE);
    }
    else
        printf("Socket creation successfull!\n");

    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // Binding newly created socket to given IP and verification
    if ((bind(sockfd, (SA *)&servaddr, sizeof(servaddr))) != 0)
    {
        fprintf(stderr, "Socket bind failed!\n");
        exit(EXIT_FAILURE);
    }
    else
        printf("Socket successfully bound\n");

    // Now server is ready to listen and verification
    if ((listen(sockfd, CLIENT)) != 0)
    {
        fprintf(stderr, "Listen failed!\n");
        exit(1);
    }
    else
        printf("Server listening for %d clients!\n", CLIENT);

    len = sizeof(cli);

    bzero(new_fd, sizeof(int) * CLIENT);
    while (1)
    {
        FD_ZERO(&readfds);
```

```c
FD_SET(sockfd, &readfds);

// Check for activity on server socket descriptor (for new connections)
int activity = select(sockfd+1, &readfds, NULL, NULL, &tv);

if (activity < 0){
    printf("Error in select()!\n");
    exit(EXIT_FAILURE);
}

// New connection detected
if (FD_ISSET(sockfd, &readfds))
{
    int client = accept(sockfd, (struct sockaddr *)&cli, &len);
    if (client < 0)
    {
        fprintf(stderr, "Accept error!\n");
        exit(1);
    }

    for (int i = 0; i < CLIENT; i++)
        if (new_fd[i] == 0)
        {
            new_fd[i] = client;
            break;
        }
    FD_CLR(sockfd, &readfds);
}


int limit = fd_max(new_fd, sockfd, &readfds);

// Query for activity on existing clients (new message obtained)
activity = select(limit, &readfds, NULL, NULL, &tv);

if (activity < 0){
    printf("Error in select()!\n");
    exit(EXIT_FAILURE);
}


for (int i = 0; i < CLIENT; i++)
{
    if (new_fd[i] < 0)
        continue;

    // Message from client
    if (FD_ISSET(new_fd[i], &readfds))
    {
        int count = read(new_fd[i], buff, MAX);

        // Client has terminated
        if (strcmp(buff, "END") == 0)
        {
            close(new_fd[i]);
            new_fd[i] = 0;
            printf("Client %d disconnected!\n", i);
        }

        else
        {
```

```c
                    printf("Client %d: %s \n", (i+1), buff);
                    bzero(buff, MAX);
                    printf("Server: ");
                    scanf("%[^\n]", buff);
                    getchar();

                    if(strcmp(buff,"KILL") == 0) exit(EXIT_SUCCESS);

                    // Write response to client
                    write(new_fd[i], buff, MAX);
                }

            }
        }
    }

    // Close the socket
    close(sockfd);
}
```

---

2. Client Side

```c
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <string.h>
#define MAX 256
#define SA struct sockaddr
int main(int argc, char **argv)
{
    if(argc < 2){
        fprintf(stderr, "Please pass port number of server as second argument!\n");
        exit(EXIT_FAILURE);
    }

    int PORT = atoi(argv[1]);
    int sockfd, connfd;
    int size;
    struct sockaddr_in servaddr, cli;
    char buff[MAX];
    int n; // socket create and varification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1)
    {
        fprintf(stderr, "Socket creation failed!\n");
        exit(0);
    }
    else
        printf("Socket creation successfull!\n");

    bzero(&servaddr, sizeof(servaddr));
    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    servaddr.sin_port = htons(PORT);

    // connect the client socket to server socket
```

```c
    if (connect(sockfd, (SA *)&servaddr, sizeof(servaddr)) != 0)
    {
        fprintf(stderr, "Connection failed!\n");
        exit(0);
    }
    else
        printf("Connection to server successfull!\n");
    while (1)
    {
        bzero(buff, MAX);
        printf("Client: ");
        scanf("%[^\n]", buff);
        getchar();
        write(sockfd, buff, MAX);

        if(strcmp(buff, "END") == 0) break;

        read(sockfd, buff, MAX);
        printf("Server: %s\n", buff);
    }
    printf("Terminated Client!\n");
    close(sockfd);
}
```

Output

Figure 1: **Server Program Output**

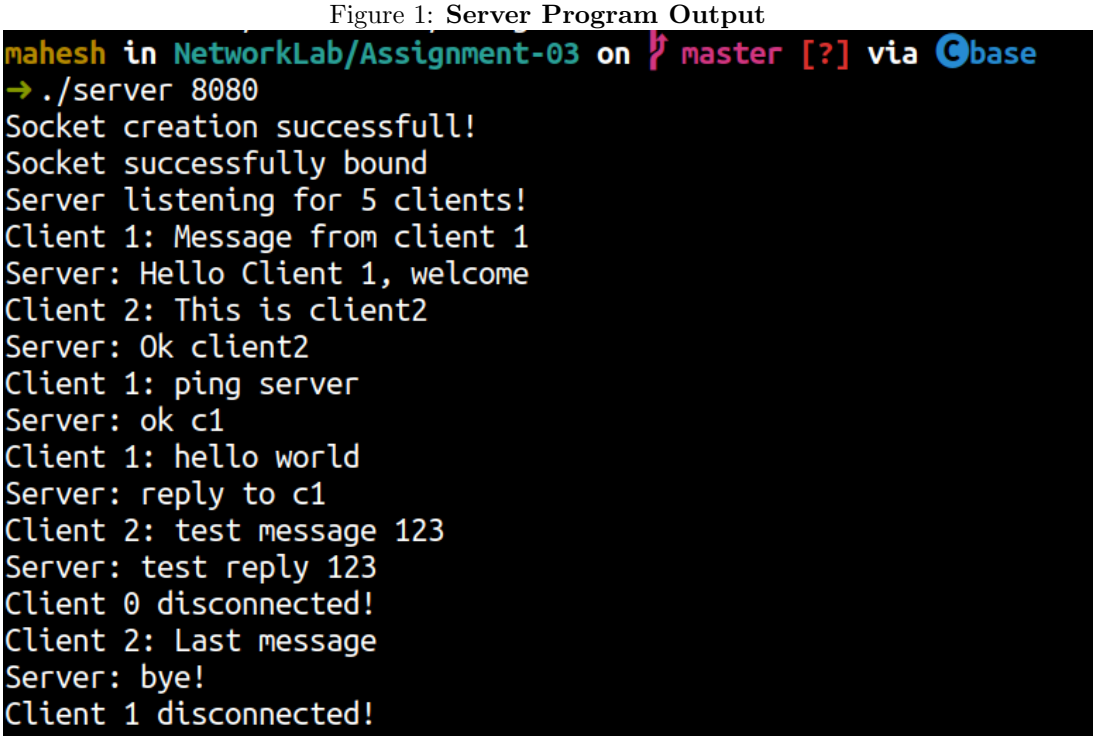

Figure 2: **Client 1 Program Output**

```
mahesh in NetworkLab/Assignment-03 on master [?] via base
➜ ./client 8080
Socket creation successfull!
Connection to server successfull!
Client: Message from client 1
Server: Hello Client 1, welcome
Client: ping server
Server: ok c1
Client: hello world
Server: reply to c1
Client: END
Terminated Client!
```

Figure 3: **Client 2 Program Output**

```
mahesh in NetworkLab/Assignment-03 on master [?] via base
➜ ./client 8080
Socket creation successfull!
Connection to server successfull!
Client: This is client2
Server: Ok client2
Client: test message 123
Server: test reply 123
Client: Last message
Server: bye!
Client: END
Terminated Client!
```

Learning Outcomes:

- We learn how to create a simple TCP client server connection.
- We learn how to appropriate system calls to set up Server and Client Programs.
- We learn to use select() to detect detect activity on file descriptors.
- We learn to handle multiple clients using a single program thread.