Distributed Systems (CS304)

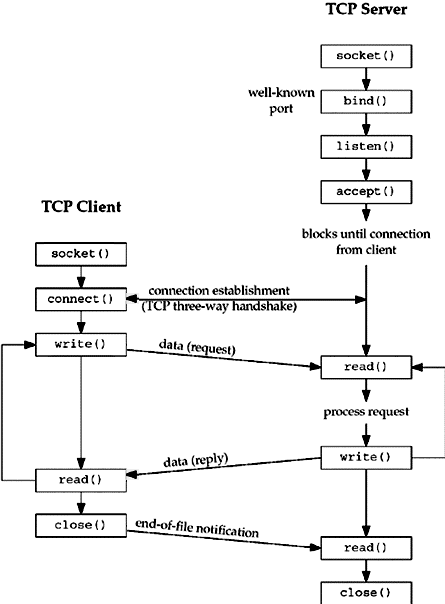
Assignment - 5

**U19CS012**

Implement given extensions to the Client Server Programming.

1. Extend your **Echo Client Server** Message Passing Application to Chat Application.

* Client and Server are able to send the message to each other until one of them **quits** or **terminates**.



2. Using the Client-Server Communication mechanism get the **Load status** of other nodes in your network (Identify the States of other nodes in the system – *Overload, Moderate, Lightly*).

* Implement the **Client-Server** model. Run the client and server instance on same machine and pass the message from client to server or server to client
* Get the CPU load of the client or server and **state** that either it is under loaded or overloaded.

[Note: The Client Server communication mechanism has the **Limitation** that it only handles **one connection** at a time and then terminates. A **real-world server** should run **indefinitely** and should have the capability of handling a number of simultaneous connections, each in its own process.]

It can be Solved using **Separate Threads** for Each Client. {Above Note}

**Code**

**[server.c]**

*#include* <stdio.h>  */\* for printf() and fprintf() \*/*

*#include* <stdlib.h> */\* for atoi() and exit() \*/*

*#include* <string.h> */\* for bzero() \*/*

*#include* <sys/types.h>

*#include* <sys/socket.h> */\* for socket(), bind(), connect(), recv() and send() \*/*

*#include* <arpa/inet.h>

*#include* <netinet/in.h>

*#include* <netdb.h>

*#include* <fcntl.h>  *// for open*

*#include* <unistd.h> *// for close*

*// Maximum Size of Buffer*

*#define* MAX 1000

*// Port Used for Socket Communication*

*#define* PORT 8080

*// Short-Hand for structure*

*#define* SA struct sockaddr

*// Message to Disconnect the Connection*

*#define* DISCONNECT\_MESSAGE "EXIT"

*// Message to Get CPU Load of System*

*#define* SYSTEM\_LOAD "CPU\_LOAD"

*// F(x) to Return the Total Load of System*

float system\_load();

*// F(x) for Communication between Server and Client*

void chat(int client\_id);

int main()

{

    int sockfd, client\_id, len;

    struct sockaddr\_in servaddr, cli;

*// socket create*

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

*if* (sockfd == -1)

    {

        printf("[+] Error : Socket Creation Failed!\n");

        exit(0);

    }

*else*

    {

        printf("[+] Socket Successfully Created!\n");

    }

    bzero(&servaddr, sizeof(servaddr));

*// Assign IP, PORT*

    servaddr.sin\_family = AF\_INET;                */\* Internet address family \*/*

    servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY); */\* Any incoming interface \*/*

    servaddr.sin\_port = htons(PORT);              */\* Local port \*/*

*// Binding newly created socket to given IP*

*if* ((bind(sockfd, (SA \*)&servaddr, sizeof(servaddr))) != 0)

    {

        printf("[+] Error : Socket Bind Failed!\n");

        exit(0);

    }

*else*

    {

        printf("[+] Socket Successfully Binded!\n");

    }

*// Now server is ready to listen*

*if* ((listen(sockfd, 5)) != 0)

    {

        printf("[+] Error : Listen Failed!\n");

        exit(0);

    }

*else*

    {

        printf("[+] Server Listening ... \n");

    }

    len = sizeof(cli);

*// Accept the data packet from client*

    client\_id = accept(sockfd, (SA \*)&cli, &len);

*if* (client\_id < 0)

    {

        printf("[+] Server Acception from Client Failed!\n");

        exit(0);

    }

*else*

    {

        printf("[+] Server Accepts the Client\n");

    }

*// Function for chatting between client and server*

    chat(client\_id);

*// Close the Socket*

    close(sockfd);

*return* 0;

}

*// F(x) to Return the Total Load of System*

float system\_load()

{

    char command[MAX], output\_of\_top\_cmd[MAX];

*// It Stores the Output in String Buffer*

    sprintf(command, "top -n1 | grep \"Cpu(s)\"");

*// Copy the Contents to "output\_of\_top\_cmd" char Array*

    FILE \*fp = popen(command, "r");

    fgets(output\_of\_top\_cmd, sizeof(output\_of\_top\_cmd), fp);

    pclose(fp);

    int token\_id = 0, i = 0;

    float Total\_Load = 0;

*// Until End of String*

*while* (output\_of\_top\_cmd[i] != '\0')

    {

        char token[MAX];

        int j = 0;

*// Until Space of EOL Character is encountered*

*while* (output\_of\_top\_cmd[i] != '\0' && output\_of\_top\_cmd[i] != ' ')

        {

            token[j] = output\_of\_top\_cmd[i], j++, i++;

        }

        token[j] = '\0';

*if* ((token\_id == 2) || (token\_id == 5))

        {

*// 2 - User CPU Usage, 5 - System Usage*

*// Convert Character Array to Float and Add*

            Total\_Load += atof(token);

        }

        token\_id++;

        i++;

    }

*return* Total\_Load;

}

*// F(x) for Communication between Server and Client*

void chat(int client\_id)

{

    int n;

    char buff[MAX];

*// Infinite loop for chat*

*while* (1)

    {

*// Clear the Buffer*

        bzero(buff, MAX);

*// Read the message from client and copy it in buffer*

        read(client\_id, buff, MAX);

*// Print buffer which contains the Client contents*

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

*if* (strncmp(SYSTEM\_LOAD, buff, 8) == 0)

        {

            bzero(buff, MAX);

            float Total\_Load = system\_load();

*if* (Total\_Load > 70)

                sprintf(buff, "CPU Load : %.2f [Overloaded]", Total\_Load);

*else* *if* (Total\_Load > 40)

                sprintf(buff, "CPU Load : %.2f [Moderate]", Total\_Load);

*else*

                sprintf(buff, "CPU Load : %.2f [Lightly]", Total\_Load);

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

            write(client\_id, buff, strlen(buff));

        }

*else* *if* (strncmp(DISCONNECT\_MESSAGE, buff, 4) == 0)

        {

            printf("\n[+] Client Disconnected!\n");

*break*;

        }

*else*

        {

            bzero(buff, MAX);

            printf("\n Enter Server Message : ");

*// Copy Server message in the buffer*

            fgets(buff, MAX, stdin);

            write(client\_id, buff, strlen(buff));

*if* (strncmp(buff, DISCONNECT\_MESSAGE, 4) == 0)

            {

                printf("\n[+] Client Disconnected!\n");

*break*;

            }

        }

    }

}

**[client.c]**

*#include* <stdio.h>      */\* for printf() and fprintf() \*/*

*#include* <arpa/inet.h>  */\* for sockaddr\_in and inet\_ntoa() \*/*

*#include* <stdlib.h>     */\* for atoi() and exit() \*/*

*#include* <string.h>     */\* for bzero() \*/*

*#include* <sys/socket.h> */\* for socket(), bind(), connect(), recv() and send() \*/*

*#include* <netdb.h>

*#include* <fcntl.h> *// for open*

*#include* <unistd.h> *// for close*

*// Maximum Size of Buffer*

*#define* MAX 1000

*// Port Used for Socket Communication*

*#define* PORT 8080

*// Short-Hand for structure*

*#define* SA struct sockaddr

*// Message to Disconnect the Connection*

*#define* DISCONNECT\_MESSAGE "EXIT"

*// Message to Get CPU Load of System*

*#define* SYSTEM\_LOAD "CPU\_LOAD"

*// F(x) to Return the Total Load of System*

float system\_load();

*// F(x) for Communication between Server and Client*

void chat(int sockfd);

int main()

{

    int sockfd, connfd;

    struct sockaddr\_in servaddr, cli;

*// socket create and verification*

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

*if* (sockfd == -1)

    {

        printf("[+] Error : Socket Creation Failed!\n");

        exit(0);

    }

*else*

    {

        printf("[+] Socket Successfully Created!\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*

*if* (connect(sockfd, (SA \*)&servaddr, sizeof(servaddr)) != 0)

    {

        printf("[+] Error : Connection with Server Failed!\n");

        exit(0);

    }

*else*

    {

        printf("[+] Connected to Server Succesfully\n");

    }

*// function for chat*

    chat(sockfd);

*// Close the socket*

    close(sockfd);

*return* 0;

}

*// F(x) to Return the Total Load of System*

float system\_load()

{

    char command[MAX], output\_of\_top\_cmd[MAX];

*// It Stores the Output in String Buffer*

    sprintf(command, "top -n1 | grep \"Cpu(s)\"");

*// Copy the Contents to "output\_of\_top\_cmd" char Array*

    FILE \*fp = popen(command, "r");

    fgets(output\_of\_top\_cmd, sizeof(output\_of\_top\_cmd), fp);

    pclose(fp);

    int token\_id = 0, i = 0;

    float Total\_Load = 0;

*// Until End of String*

*while* (output\_of\_top\_cmd[i] != '\0')

    {

        char token[MAX];

        int j = 0;

*// Until Space of EOL Character is encountered*

*while* (output\_of\_top\_cmd[i] != '\0' && output\_of\_top\_cmd[i] != ' ')

        {

            token[j] = output\_of\_top\_cmd[i], j++, i++;

        }

        token[j] = '\0';

*if* ((token\_id == 2) || (token\_id == 5))

        {

*// 2 - User CPU Usage, 5 - System Usage*

*// Convert Character Array to Float and Add*

            Total\_Load += atof(token);

        }

        token\_id++;

        i++;

    }

*return* Total\_Load;

}

*// F(x) for Communication between Server and Client*

void chat(int sockfd)

{

    int n;

    char buff[MAX];

*// Infinite loop for chat*

*while* (1)

    {

*if* (strncmp(SYSTEM\_LOAD, buff, 8) == 0)

        {

            bzero(buff, MAX);

            float Total\_Load = system\_load();

*if* (Total\_Load > 70)

                sprintf(buff, "CPU Load : %.2f [Overloaded]", Total\_Load);

*else* *if* (Total\_Load > 40)

                sprintf(buff, "CPU Load : %.2f [Moderate]", Total\_Load);

*else*

                sprintf(buff, "CPU Load : %.2f [Lightly]", Total\_Load);

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

            write(sockfd, buff, strlen(buff));

        }

*else* *if* (strncmp(DISCONNECT\_MESSAGE, buff, 4) == 0)

        {

            printf("\n[+] Server Disconnected!\n");

*break*;

        }

*else*

        {

            bzero(buff, MAX);

            printf("\n Enter Client Message : ");

            fgets(buff, MAX, stdin);

            write(sockfd, buff, strlen(buff));

*if* (strncmp(buff, DISCONNECT\_MESSAGE, 4) == 0)

            {

                printf("\n[+] Client Disconnected!\n");

*break*;

            }

        }

        bzero(buff, MAX);

        read(sockfd, buff, MAX);

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

    }

}

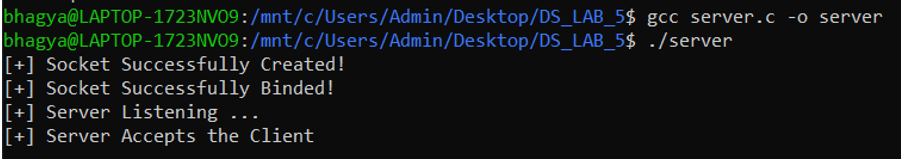
**Output**

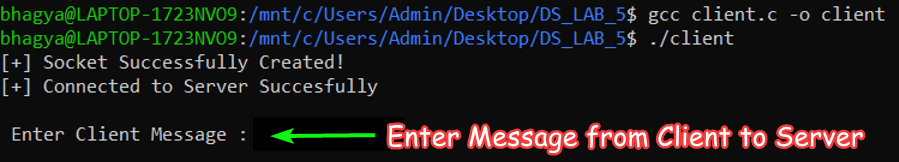
**Step 1**: Compile both server.c and client.c to generate the executable Files.

Start the **Server by executing** the server.exe

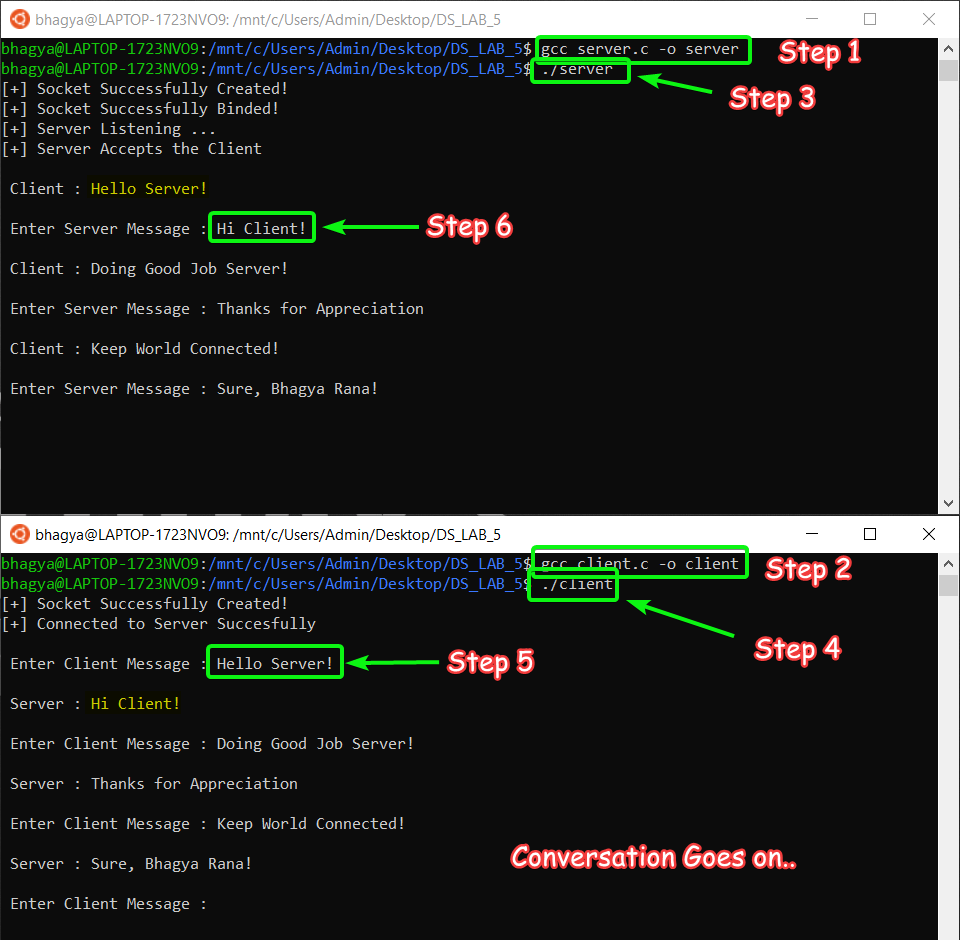


**Step 2**: Run the Client, So Server gets the Client Connected and Ready to **Chat** with Server.





**Step 5**: Two Way Messaging can be done!



**SUBMITTED BY**: U19CS012

BHAGYA VINOD RANA