Distributed Systems (CS304)

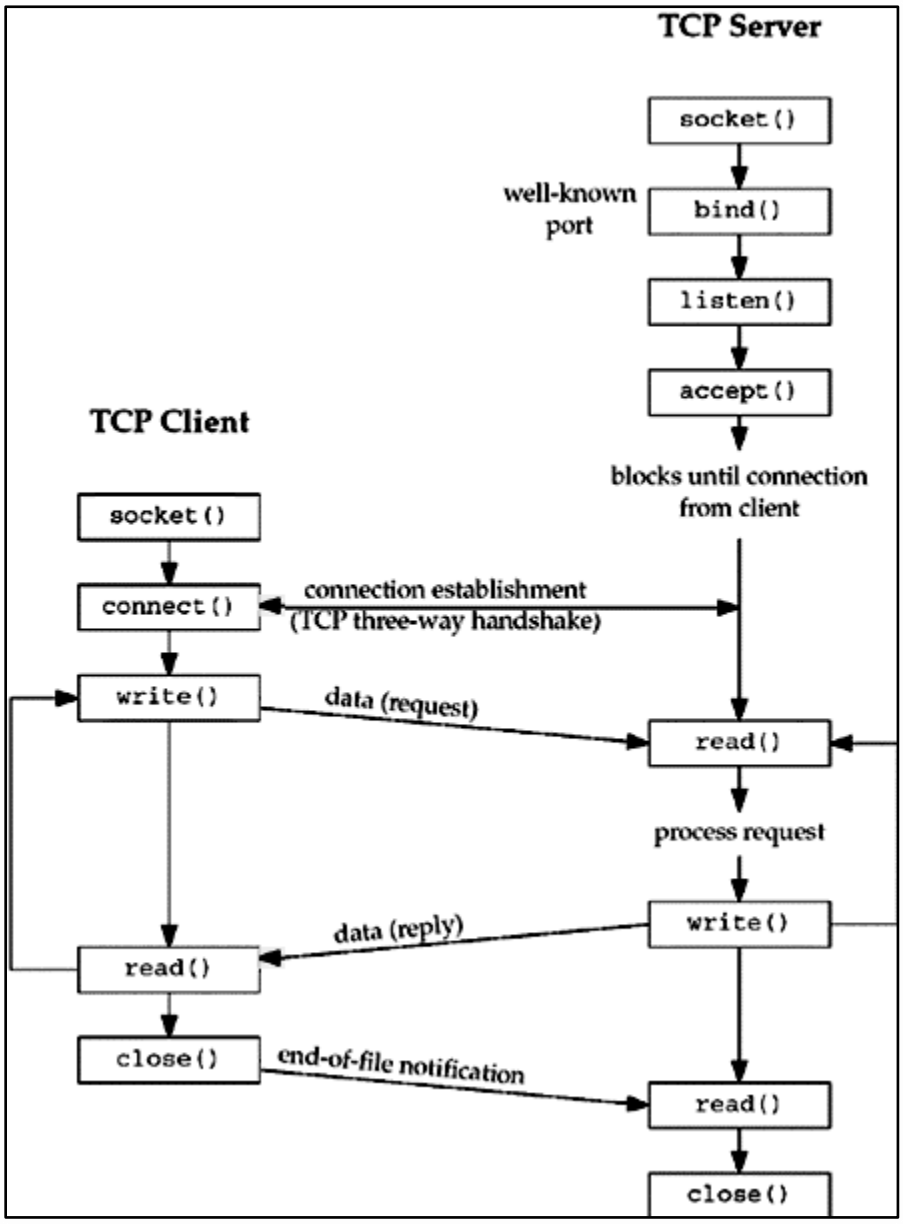
Assignment - 5

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();
```

```c
// 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);
```

```c
    // 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';
```

```c
        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;
        }
```

```c
        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]

```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;
```

```c
    // 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`

```
bhagya@LAPTOP-1723NVO9: /mnt/c/Users/Admin/Desktop/DS_LAB_4
bhagya@LAPTOP-1723NVO9:/mnt/c/Users/Admin/Desktop/DS_LAB_4$ g++ -o server.exe server.c
bhagya@LAPTOP-1723NVO9:/mnt/c/Users/Admin/Desktop/DS_LAB_4$ ./server.exe
Socket Created!
Bind Done!
Waiting for Incoming Clients Connections ...
```

Step 1

Step 3

Server Waiting for Client to Connect

```
bhagya@LAPTOP-1723NVO9: /mnt/c/Users/Admin/Desktop/DS_LAB_4
bhagya@LAPTOP-1723NVO9:/mnt/c/Users/Admin/Desktop/DS_LAB_4$ g++ -o client.exe client.c
bhagya@LAPTOP-1723NVO9:/mnt/c/Users/Admin/Desktop/DS_LAB_4$
```

Step 2

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

```
bhagya@LAPTOP-1723NVO9:/mnt/c/Users/Admin/Desktop/DS_LAB_5$ gcc server.c -o server
bhagya@LAPTOP-1723NVO9:/mnt/c/Users/Admin/Desktop/DS_LAB_5$ ./server
[+] Socket Successfully Created!
[+] Socket Successfully Binded!
[+] Server Listening ...
[+] Server Accepts the Client
```

```
bhagya@LAPTOP-1723NVO9:/mnt/c/Users/Admin/Desktop/DS_LAB_5$ gcc client.c -o client
bhagya@LAPTOP-1723NVO9:/mnt/c/Users/Admin/Desktop/DS_LAB_5$ ./client
[+] Socket Successfully Created!
[+] Connected to Server Succesfully

Enter Client Message :
```

Enter Message from Client to Server

Step 5: Two Way Messaging can be done!

```
bhagya@LAPTOP-1723NVO9: /mnt/c/Users/Admin/Desktop/DS_LAB_5
bhagya@LAPTOP-1723NVO9:/mnt/c/Users/Admin/Desktop/DS_LAB_5$ gcc server.c -o server      Step 1
bhagya@LAPTOP-1723NVO9:/mnt/c/Users/Admin/Desktop/DS_LAB_5$ ./server      Step 3
[+] Socket Successfully Created!
[+] Socket Successfully Binded!
[+] Server Listening ...
[+] Server Accepts the Client

 Client : Hello Server!

 Enter Server Message : Hi Client!      Step 6

 Client : Doing Good Job Server!

 Enter Server Message : Thanks for Appreciation

 Client : Keep World Connected!

 Enter Server Message : Sure, Bhagya Rana!
```

```
bhagya@LAPTOP-1723NVO9: /mnt/c/Users/Admin/Desktop/DS_LAB_5
bhagya@LAPTOP-1723NVO9:/mnt/c/Users/Admin/Desktop/DS_LAB_5$ gcc client.c -o client      Step 2
bhagya@LAPTOP-1723NVO9:/mnt/c/Users/Admin/Desktop/DS_LAB_5$ ./client      Step 4
[+] Socket Successfully Created!
[+] Connected to Server Succesfully

 Enter Client Message : Hello Server!      Step 5

 Server : Hi Client!

 Enter Client Message : Doing Good Job Server!

 Server : Thanks for Appreciation

 Enter Client Message : Keep World Connected!

 Server : Sure, Bhagya Rana!      Conversation Goes on..

 Enter Client Message :
```

SUBMITTED BY: U19CS012

BHAGYA VINOD RANA