Assignment 4

Implement given extensions to the Client Server Programming.

1. Extend your echo Client Server massage passing application to chat application. Client and Server able to send the message to each other until one of them quit or terminate.

2. Using 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 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 client or server and state that either it is under loaded or overloaded.

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.

Code for client:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

int main()
{
    char *ip = "127.0.0.1";
    int port = 5000;

    int sock;
    struct sockaddr_in addr;
    socklen_t addr_size;
    char buffer[1024];
    int n;

    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (n < 0)
    {
        perror("Socket error.....");

printf("#########################################################\n");
        printf("\n");
        exit(1);
    }
```

```
    printf("TCP server socket created......\n");

    memset(&addr, '\0', sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = port;
    addr.sin_addr.s_addr = inet_addr(ip);

    connect(sock, (struct sockaddr *)&addr, sizeof(addr));
    printf("Server connected......\n");

printf("#########################################################\n");
    printf("\n");

    while (1)
    {

        bzero(buffer, 1024);
        printf("Enter message to send a server (exit for quit, load for
get load) : \n");
        scanf("%[^\n]%*c", buffer);

        printf("Your message sended to server is : \n");
        printf("%s", buffer);
        printf("\n\n");

        send(sock, buffer, strlen(buffer), 0);

        if ((strncmp(buffer, "exit", 4)) == 0)
        {
            close(sock);
            printf("Server disconnected......\n");

printf("#########################################################\n");
            printf("\n");
            exit(0);
        }

    receive:
        bzero(buffer, 1024);
        recv(sock, buffer, sizeof(buffer), 0);
        printf("Message from server is : %s\n", buffer);
        printf("\n");

        if ((strncmp(buffer, "exit", 4)) == 0)
        {
            close(sock);
            printf("Server disconnected......\n");
```

```
printf("#########################################################\n");
            printf("\n");
            exit(0);
        }

        if ((strncmp(buffer, "load", 4)) == 0)
        {
            bzero(buffer, 1024);
            system("mpstat| grep -w \"all\"| awk '{o = 100-$NF; {print
o} }' >> load.txt");
            FILE *file;
            char load[10];
            float cpu_load;
            file = fopen("load.txt", "r");
            if (file == NULL)
            {
                printf("file can't be opened......\n");

printf("#########################################################\n");
                printf("\n");
            }

            fgets(load, 10, file);
            fclose(file);
            cpu_load = atof(load);
            if (cpu_load > 70)
            {
                strcpy(buffer, "Client is overload\n");
            }
            else if (30 > cpu_load && 70 < cpu_load)
            {
                strcpy(buffer, "Client is moderately loaded\n");
            }
            else
            {
                strcpy(buffer, "Client is lightly loaded\n");
            }

            printf("Your message sended to server is : \n");
            printf("%s", buffer);
            printf("\n\n");
            send(sock, buffer, strlen(buffer), 0);
            goto receive;
        }
    }

    return 0;
```

```
}
```

Code for server:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

int main()
{
    char *ip = "127.0.0.1";
    int port = 5000;

    int server_sock, client_sock;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_size;
    char buffer[1024];
    int n;

    server_sock = socket(AF_INET, SOCK_STREAM, 0);
    if (server_sock < 0)
    {
        perror("Socket error......\n");

printf("#########################################################\n");
        printf("\n");
        exit(1);
    }

    memset(&server_addr, '\0', sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = port;
    server_addr.sin_addr.s_addr = inet_addr(ip);

    n = bind(server_sock, (struct sockaddr *)&server_addr,
sizeof(server_addr));
    if (n < 0)
    {
        perror("Bind error......");

printf("#########################################################\n");
        printf("\n");
        exit(1);
    }

    listen(server_sock, 5);
    printf("Serever running......\n");
```

```
printf("#########################################################\n");
    printf("\n");

    addr_size = sizeof(client_addr);
    client_sock = accept(server_sock, (struct sockaddr *)&client_addr,
&addr_size);
    printf("Client Connected......\n");

printf("#########################################################\n");
    printf("\n");

    while (1)
    {
        bzero(buffer, 1024);
        recv(client_sock, buffer, sizeof(buffer), 0);
        printf("Message from client is : %s\n", buffer);
        printf("\n");

        if ((strncmp(buffer, "exit", 4)) == 0)
        {
            // When client send exit request
            close(client_sock);
            printf("Client disconnected......\n");

printf("#########################################################\n");
            printf("\n");

            // For client to enter server again so not stuck here
            client_sock = accept(server_sock, (struct sockaddr
*)&client_addr, &addr_size);
            printf("Client Connected......\n");

printf("#########################################################\n");
            printf("\n");
        }

        if ((strncmp(buffer, "load", 4)) == 0)
        {
            bzero(buffer, 1024);
            printf("Send message to client %d : ", client_sock);
            system("mpstat| grep -w \"all\"| awk '{o = 100-$NF; {print
o} }' >> load.txt");
            FILE *file;
            char load[10];
            float cpu_load;
            file = fopen("load.txt", "r");
            if (file == NULL)
```

```
            {
                printf("file can't be opened......\n");

printf("#########################################################\n");
                printf("\n");
            }

            fgets(load, 10, file);
            fclose(file);
            cpu_load = atof(load);
            if (cpu_load > 70)
            {
                strcpy(buffer, "Server is overload\n");
            }
            else if (30 > cpu_load && 70 < cpu_load)
            {
                strcpy(buffer, "Server is moderately loaded\n");
            }
            else
            {
                strcpy(buffer, "Server is lightly loaded\n");
            }

            printf("Your message sended to client is : \n");
            printf("%s", buffer);
            printf("\n\n");
            send(client_sock, buffer, strlen(buffer), 0);

            continue;
        }

        bzero(buffer, 1024);
        printf("Enter message to send client %d (exit for quit, load
for get load) : ", client_sock);
        scanf("%[^\n]%*c", buffer);

        printf("Your message sended to client is : \n");
        printf("%s", buffer);
        printf("\n\n");
        send(client_sock, buffer, strlen(buffer), 0);

        if ((strncmp(buffer, "exit", 4)) == 0)
        {
            close(client_sock);
            printf("Client disconnected......\n");

printf("#########################################################\n");
            printf("\n");
```

```
            client_sock = accept(server_sock, (struct sockaddr
*)&client_addr, &addr_size);
            printf("Client Connected......\n");

printf("#########################################################\n");
            printf("\n");
        }
    }

    return 0;
}
```

Output for client:

```
┌──(nihar㉿nihar)-[~/…/Sem 2/Distributed Systems/LABS/LAB 4]
└─$ ./client
TCP server socket created......
Server connected......
#########################################################

Enter message to send a server (exit for quit, load for get load) :
hi
Your message sended to server is :
hi

Message from server is : hi

Enter message to send a server (exit for quit, load for get load) :
hi
Your message sended to server is :
hi

Message from server is : exit

Server disconnected......
#########################################################
```

Output for server:

```
┌──(nihar㉿nihar)-[~/…/Sem 2/Distributed Systems/LABS/LAB 4]
└─$ gcc server.c -o server

┌──(nihar㉿nihar)-[~/…/Sem 2/Distributed Systems/LABS/LAB 4]
└─$ ./server
Serever running......
#########################################################

Client Connected......
#########################################################

Message from client is : hi

Enter message to send client 4 (exit for quit, load for get load) : hi
Your message sended to client is :
hi

Message from client is : hi

Enter message to send client 4 (exit for quit, load for get load) : exit
Your message sended to client is :
exit

Client disconnected......
#########################################################
```