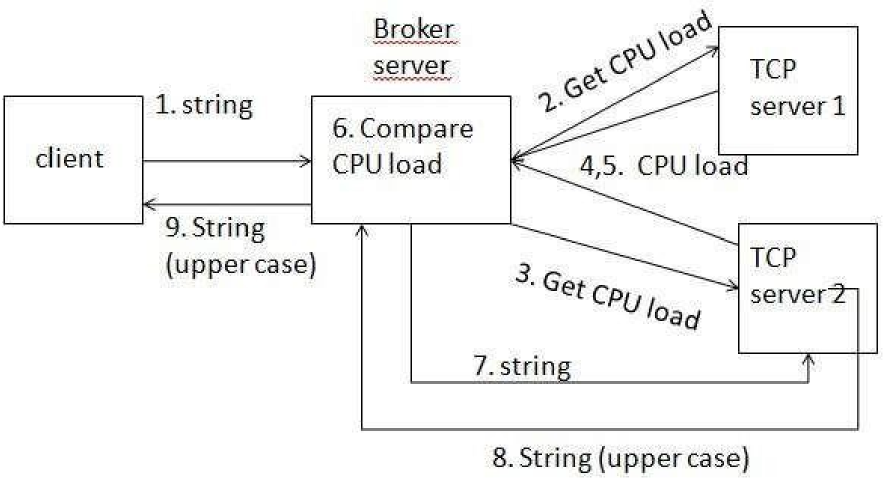
Name -Sumit Kumar Jha
Reg -20204212
Sec - CSE C

Motilal Nehru National Institute of Technology Allahabad Prayagraj
Distributed System (CS17201)
B.Tech (CSE) – VII Sem Lab 6

Q1. Suppose you have two TCP servers for converting a lower case string to upper case string. You have to design a load balancer server that accepts lower case strings from the client and check for the CPU utilization of both servers. Load balancer will transfer the string to the server having less CPU utilization. The load balancer will get upper case string from server and return to the clients.
Code :



Load_balancer.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#define LOAD_BALANCER_PORT 8888
#define SERVER_COUNT 2
#define SERVER1_IP "127.0.0.1"
#define SERVER1_PORT 8889
#define SERVER2_IP "127.0.0.1"
#define SERVER2_PORT 8890

int main()
{
    int loadBalancerSocket, serverSockets[SERVER_COUNT];
    struct sockaddr_in loadBalancerAddr, serverAddrs[SERVER_COUNT];

    // Create a socket for the load balancer
    loadBalancerSocket = socket(AF_INET, SOCK_STREAM, 0);

    // Initialize the load balancer address structure
    memset(&loadBalancerAddr, 0, sizeof(loadBalancerAddr));
    loadBalancerAddr.sin_family = AF_INET;
    loadBalancerAddr.sin_addr.s_addr = INADDR_ANY;
    loadBalancerAddr.sin_port = htons(LOAD_BALANCER_PORT);

    // Bind the load balancer socket
    bind(loadBalancerSocket, (struct sockaddr *)&loadBalancerAddr,
sizeof(loadBalancerAddr));

    // Listen for incoming connections
    listen(loadBalancerSocket, SERVER_COUNT);

    // Create sockets for Server 1 and Server 2
    serverSockets[0] = socket(AF_INET, SOCK_STREAM, 0);
    serverSockets[1] = socket(AF_INET, SOCK_STREAM, 0);

    // Initialize server addresses
    memset(&serverAddrs[0], 0, sizeof(serverAddrs[0]));
    serverAddrs[0].sin_family = AF_INET;
    serverAddrs[0].sin_addr.s_addr = inet_addr(SERVER1_IP);
    serverAddrs[0].sin_port = htons(SERVER1_PORT);
    memset(&serverAddrs[1], 0, sizeof(serverAddrs[1]));
    serverAddrs[1].sin_family = AF_INET;
    serverAddrs[1].sin_addr.s_addr = inet_addr(SERVER2_IP);
    serverAddrs[1].sin_port = htons(SERVER2_PORT);

    // Connect to Server 1 and Server 2
    connect(serverSockets[0], (struct sockaddr *)&serverAddrs[0],
sizeof(serverAddrs[0]));
    connect(serverSockets[1], (struct sockaddr *)&serverAddrs[1],
sizeof(serverAddrs[1]));
    int currentServer = 0; // Variable to keep track of the selected server
    while (1)
    {
        int clientSocket;

        // Accept an incoming connection from a client
        clientSocket = accept(loadBalancerSocket, NULL, NULL);
        printf("Accepted connection from a client.\n");
        char buffer[1024];
        ssize_t bytesRead;

        // Read the message from the client
        bytesRead = read(clientSocket, buffer, sizeof(buffer));
        printf("Received from client: %s\n", buffer);

        // Forward the message to the selected server
        write(serverSockets[currentServer], buffer, bytesRead);
        printf("Forwarded to server %d: %s\n", currentServer + 1, buffer);
        char serverResponse[1024]; // Response buffer for server response
        ssize_t serverResponseBytes;

        // Receive the response from the server
        serverResponseBytes = read(serverSockets[currentServer],
serverResponse,sizeof(serverResponse));
        printf("Received from server %d: %s\n", currentServer + 1,
serverResponse);

        // Send the response back to the client
        write(clientSocket, serverResponse, serverResponseBytes);
        printf("Sent response to client: %s\n", serverResponse);

        // Close the client socket
        close(clientSocket);

        // Switch to the other server in a round-robin fashion
        currentServer = (currentServer + 1) % SERVER_COUNT;
    }

    // Close sockets and clean up (not reached in this simplified example)
    close(loadBalancerSocket);
    close(serverSockets[0]);
    close(serverSockets[1]);
    return 0;
}
```

Server1.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <ctype.h>
#define SERVER1_PORT 8889
int main() {
    int serverSocket;
    struct sockaddr_in serverAddr;

    // Create a socket for Server 1
    serverSocket = socket(AF_INET, SOCK_STREAM, 0);

    // Initialize the server address structure
    memset(&serverAddr, 0, sizeof(serverAddr));
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(SERVER1_PORT);

    // Bind the server socket
    bind(serverSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr));

    // Listen for incoming connections
    listen(serverSocket, 5);
    while (1) {
    int clientSocket;
    char buffer[1024];
    ssize_t bytesRead;

    // Accept an incoming connection from a load balancer
    clientSocket = accept(serverSocket, NULL, NULL);
    printf("Accepted connection from the load balancer.\n");

    // Receive the message from the load balancer
    bytesRead = read(clientSocket, buffer, sizeof(buffer));
    printf("Received from the load balancer: %s\n", buffer);

    // Process the message (e.g., convert to uppercase in this example)
    for (int i = 0; i < bytesRead; i++) {
    buffer[i] = toupper(buffer[i]);
    }

    // Send the processed message back to the load balancer
    write(clientSocket, buffer, bytesRead);
    printf("Processed message: %s\n", buffer);

    // Close the client socket
    close(clientSocket);
    }

    // Close the server socket (not reached in this simplified example)
    close(serverSocket);
    return 0;
}
```

Server2.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <ctype.h>
#define SERVER2_PORT 8890
int main()
{
    int serverSocket;
    struct sockaddr_in serverAddr;

    // Create a socket for Server 2
```

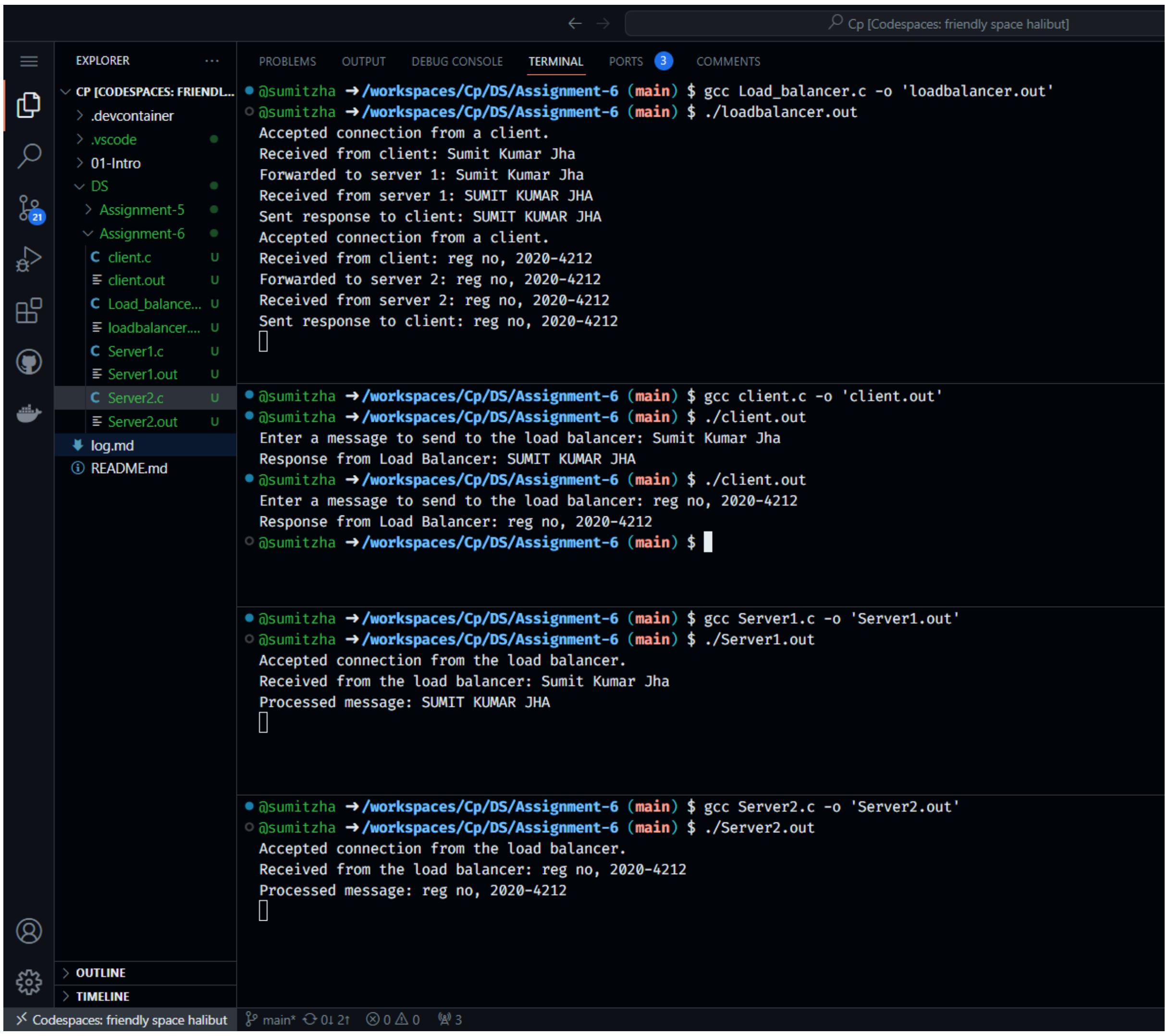
```c
serverSocket = socket(AF_INET, SOCK_STREAM, 0);

// Initialize the server address structure
memset(&serverAddr, 0, sizeof(serverAddr));
serverAddr.sin_family = AF_INET;
serverAddr.sin_addr.s_addr = INADDR_ANY;
serverAddr.sin_port = htons(SERVER2_PORT);

// Bind the server socket
bind(serverSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr));

// Listen for incoming connections
listen(serverSocket, 5);
while (1)
{
    int clientSocket;
    char buffer[1024];
    ssize_t bytesRead;

    // Accept an incoming connection from a load balancer
    clientSocket = accept(serverSocket, NULL, NULL);
    printf("Accepted connection from the load balancer.\n");

    // Receive the message from the load balancer
    bytesRead = read(clientSocket, buffer, sizeof(buffer));
    printf("Received from the load balancer: %s\n", buffer);

    // Process the message (e.g., convert to lowercase in this example)
    for (int i = 0; i < bytesRead; i++)
    {
        buffer[i] = tolower(buffer[i]);
    }

    // Send the processed message back to the load balancer
    write(clientSocket, buffer, bytesRead);
    printf("Processed message: %s\n", buffer);

    // Close the client socket
```

```
        close(clientSocket);
    }

    // Close the server socket (not reached in this simplified example)
    close(serverSocket);
    return 0;
}
```

Client.c

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

int main()
{
    int clientSocket;
    struct sockaddr_in serverAddr;

    // Create a socket for the client
    clientSocket = socket(AF_INET, SOCK_STREAM, 0);

    // Initialize the server address structure
    memset(&serverAddr, 0, sizeof(serverAddr));
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1"); // Change to the load
balancer'sIP
    serverAddr.sin_port = htons(8888);                   // Use the load
balancer's port

    // Connect to the load balancer
    connect(clientSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr));
    char message[1024];
    printf("Enter a message to send to the load balancer: ");
    fgets(message, sizeof(message), stdin);
    message[strcspn(message, "\n")] = '\0'; // Remove the newline character

    // Send the message to the load balancer
```

```c
    write(clientSocket, message, strlen(message));
    char buffer[1024];
    ssize_t bytesRead;

    // Receive the response from the load balancer
    bytesRead = read(clientSocket, buffer, sizeof(buffer));
    printf("Response from Load Balancer: %s\n", buffer);

    // Close the client socket
    close(clientSocket);
    return 0;
}
```



```
@sumitzha ➜ /workspaces/Cp/DS/Assignment-6 (main) $ gcc Load_balancer.c -o 'loadbalancer.out'
@sumitzha ➜ /workspaces/Cp/DS/Assignment-6 (main) $ ./loadbalancer.out
Accepted connection from a client.
Received from client: Sumit Kumar Jha
Forwarded to server 1: Sumit Kumar Jha
Received from server 1: SUMIT KUMAR JHA
Sent response to client: SUMIT KUMAR JHA
Accepted connection from a client.
Received from client: reg no, 2020-4212
Forwarded to server 2: reg no, 2020-4212
Received from server 2: reg no, 2020-4212
Sent response to client: reg no, 2020-4212
```

```
@sumitzha ➜ /workspaces/Cp/DS/Assignment-6 (main) $ gcc client.c -o 'client.out'
@sumitzha ➜ /workspaces/Cp/DS/Assignment-6 (main) $ ./client.out
Enter a message to send to the load balancer: Sumit Kumar Jha
Response from Load Balancer: SUMIT KUMAR JHA
@sumitzha ➜ /workspaces/Cp/DS/Assignment-6 (main) $ ./client.out
Enter a message to send to the load balancer: reg no, 2020-4212
Response from Load Balancer: reg no, 2020-4212
@sumitzha ➜ /workspaces/Cp/DS/Assignment-6 (main) $
```

```
@sumitzha ➜ /workspaces/Cp/DS/Assignment-6 (main) $ gcc Server1.c -o 'Server1.out'
@sumitzha ➜ /workspaces/Cp/DS/Assignment-6 (main) $ ./Server1.out
Accepted connection from the load balancer.
Received from the load balancer: Sumit Kumar Jha
Processed message: SUMIT KUMAR JHA
```

```
@sumitzha ➜ /workspaces/Cp/DS/Assignment-6 (main) $ gcc Server2.c -o 'Server2.out'
@sumitzha ➜ /workspaces/Cp/DS/Assignment-6 (main) $ ./Server2.out
Accepted connection from the load balancer.
Received from the load balancer: reg no, 2020-4212
Processed message: reg no, 2020-4212
```