**1. TCP Server Code (server1.c and server2.c)**

Each server will listen for a connection and either return the CPU load or perform the string conversion.

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <ctype.h>

#define PORT 12345 // Use a different port for server2 (e.g., 12346)

#define BUFFER\_SIZE 1024

// Function to calculate CPU usage (dummy value for simplicity)

float get\_cpu\_load() {

return (rand() % 100) / 100.0; // Return a random CPU usage between 0 and 1

}

// Function to convert a string to uppercase

void to\_uppercase(char \*str) {

for (int i = 0; str[i]; i++) {

str[i] = toupper(str[i]);

}

}

int main() {

int server\_socket, client\_socket;

struct sockaddr\_in server\_addr, client\_addr;

char buffer[BUFFER\_SIZE];

socklen\_t addr\_len;

// Create socket

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

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

// Bind the socket to an address

bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

// Start listening for connections

listen(server\_socket, 5);

printf("Server listening on port %d...\n", PORT);

while (1) {

addr\_len = sizeof(client\_addr);

client\_socket = accept(server\_socket, (struct sockaddr \*)&client\_addr, &addr\_len);

printf("Accepted connection from load balancer...\n");

// Receive command (either "CPU" for load or string for conversion)

recv(client\_socket, buffer, BUFFER\_SIZE, 0);

buffer[strlen(buffer)] = '\0'; // Ensure null termination

if (strcmp(buffer, "CPU") == 0) {

// Send CPU load

float cpu\_load = get\_cpu\_load();

send(client\_socket, &cpu\_load, sizeof(cpu\_load), 0);

printf("Sent CPU load: %.2f\n", cpu\_load);

} else {

// Convert string to uppercase

to\_uppercase(buffer);

send(client\_socket, buffer, strlen(buffer), 0);

printf("Sent uppercase string: %s\n", buffer);

}

close(client\_socket);

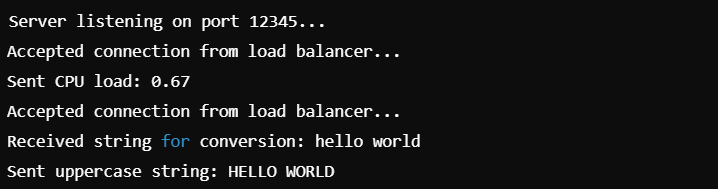
}

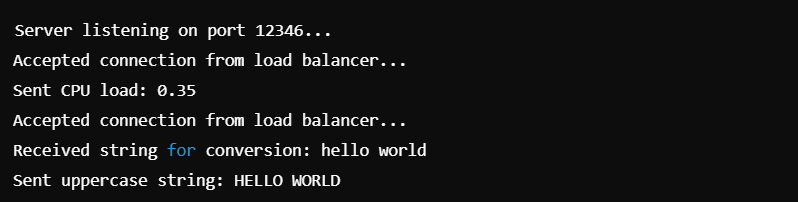
close(server\_socket);

return 0;

}

Output:





**2. Load Balancer Code (broker.c)**

The load balancer will handle client requests, compare the CPU usage of the two servers, forward the string to the least-loaded server, and return the result to the client.

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <arpa/inet.h>

#define SERVER1\_PORT 12345

#define SERVER2\_PORT 12346

#define BUFFER\_SIZE 1024

// Function to query CPU load from a server

float get\_cpu\_load(int port) {

int sock;

struct sockaddr\_in server\_addr;

float cpu\_load;

// Create socket

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

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(port);

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

// Connect to the server

connect(sock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

// Send "CPU" request to get CPU load

send(sock, "CPU", 3, 0);

recv(sock, &cpu\_load, sizeof(cpu\_load), 0);

close(sock);

return cpu\_load;

}

// Function to send a string to a server for conversion

void send\_string\_to\_server(int port, char \*str, char \*result) {

int sock;

struct sockaddr\_in server\_addr;

// Create socket

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

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(port);

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

// Connect to the server

connect(sock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

// Send the string for conversion

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

recv(sock, result, BUFFER\_SIZE, 0);

close(sock);

}

int main() {

int broker\_socket, client\_socket;

struct sockaddr\_in broker\_addr, client\_addr;

char buffer[BUFFER\_SIZE], result[BUFFER\_SIZE];

socklen\_t addr\_len;

// Create socket for load balancer

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

broker\_addr.sin\_family = AF\_INET;

broker\_addr.sin\_port = htons(12347); // Broker server port

broker\_addr.sin\_addr.s\_addr = INADDR\_ANY;

// Bind the socket

bind(broker\_socket, (struct sockaddr \*)&broker\_addr, sizeof(broker\_addr));

listen(broker\_socket, 5);

printf("Load Balancer listening on port 12347...\n");

while (1) {

addr\_len = sizeof(client\_addr);

client\_socket = accept(broker\_socket, (struct sockaddr \*)&client\_addr, &addr\_len);

printf("Accepted connection from client...\n");

// Receive string from the client

recv(client\_socket, buffer, BUFFER\_SIZE, 0);

buffer[strlen(buffer)] = '\0'; // Ensure null termination

// Get CPU loads from both servers

float cpu1 = get\_cpu\_load(SERVER1\_PORT);

float cpu2 = get\_cpu\_load(SERVER2\_PORT);

printf("CPU load - Server 1: %.2f, Server 2: %.2f\n", cpu1, cpu2);

// Send the string to the server with the lower CPU load

if (cpu1 < cpu2) {

send\_string\_to\_server(SERVER1\_PORT, buffer, result);

} else {

send\_string\_to\_server(SERVER2\_PORT, buffer, result);

}

// Send the result back to the client

send(client\_socket, result, strlen(result), 0);

printf("Sent result back to client: %s\n", result);

close(client\_socket);

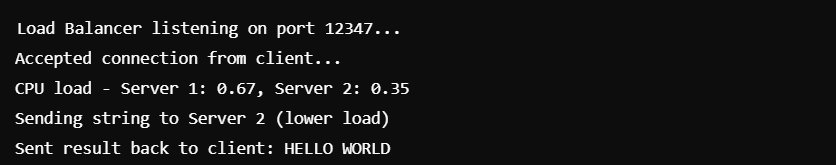
}

close(broker\_socket);

return 0;

}

Output:



**3. Client Code (client.c)**

The client sends a string to the load balancer and receives the converted string back.

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <arpa/inet.h>

#define BROKER\_PORT 12347

#define BUFFER\_SIZE 1024

int main() {

int sock;

struct sockaddr\_in broker\_addr;

char buffer[BUFFER\_SIZE], result[BUFFER\_SIZE];

// Create socket

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

broker\_addr.sin\_family = AF\_INET;

broker\_addr.sin\_port = htons(BROKER\_PORT);

broker\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

// Connect to load balancer

connect(sock, (struct sockaddr \*)&broker\_addr, sizeof(broker\_addr));

// Input string to send

printf("Enter a string to convert: ");

fgets(buffer, BUFFER\_SIZE, stdin);

buffer[strlen(buffer) - 1] = '\0'; // Remove newline

// Send string to load balancer

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

// Receive the uppercase string

recv(sock, result, BUFFER\_SIZE, 0);

printf("Received uppercase string: %s\n", result);

close(sock);

return 0;

}

Output:

