SUNNY KUMAR
26TH JUNE TASK
LSP (LINUX SYSTEM PROGRAMMING)

```
#include <iostream>
#include <thread>
#include <vector>
#include <atomic>
#include <csignal>
#include <cstring>
#include <mutex>
#include <netinet/in.h>
#include <sys/socket.h>
#include <unistd.h>

std::atomic<bool> keep_running(true);
std::mutex cout_mutex;

void signal_handler(int signal) {
    keep_running = false;
}

void handle_client(int client_socket) {
    char buffer[1024];
    while (keep_running) {
        ssize_t bytes_received = recv(client_socket, buffer, sizeof(buffer), 0);
        if (bytes_received <= 0) {
            break;
        }

        send(client_socket, buffer, bytes_received, 0);

        std::lock_guard<std::mutex> lock(cout_mutex);
        std::cout << "Received and sent back: " << std::string(buffer, bytes_received) << std::endl;
    }
    close(client_socket);
}

int main(int argc, char* argv[]) {
    if (argc != 2) {
        std::cerr << "Usage: server <port>\n";
        return 1;
    }
```

```
    int port = std::atoi(argv[1]);

    // Setup signal handling
    std::signal(SIGINT, signal_handler);
    std::signal(SIGTERM, signal_handler);

    // Create a socket
    int server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == -1) {
        std::cerr << "Failed to create socket\n";
        return 1;
    }

    // Bind the socket to the specified port
    sockaddr_in server_addr = {};
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(port);

    if (bind(server_socket, (struct sockaddr*)&server_addr, sizeof(server_addr)) == -1) {
        std::cerr << "Failed to bind socket\n";
        close(server_socket);
        return 1;
    }

    // Start listening for connections
    if (listen(server_socket, SOMAXCONN) == -1) {
        std::cerr << "Failed to listen on socket\n";
        close(server_socket);
        return 1;
    }

    std::cout << "Server is listening on port " << port << "\n";

    std::vector<std::thread> client_threads;

    while (keep_running) {
        // Accept new connections
        sockaddr_in client_addr = {};
        socklen_t client_addr_size = sizeof(client_addr);
        int client_socket = accept(server_socket, (struct sockaddr*)&client_addr,
&client_addr_size);
```

```
        if (client_socket == -1) {
            if (keep_running) {
                std::cerr << "Failed to accept connection\n";
            }
            continue;
        }

        // Handle the client connection in a new thread
        client_threads.emplace_back(handle_client, client_socket);
    }

    // Wait for all client threads to finish
    for (auto& thread : client_threads) {
        if (thread.joinable()) {
            thread.join();
        }
    }

    close(server_socket);
    std::cout << "Server has shut down gracefully.\n";
    return 0;
}
```