Task day 26

1)Design and implement a network service that reliably handles concurrent client connections while ensuring graceful termination in response to external signals (e.g., SIGTERM, SIGINT). The service must maintain data consistency and avoid resource leaks throughout its lifecycle.

Server:

#include <iostream>

#include <thread>

#include <vector>

#include <mutex>

#include <atomic>

#include <csignal>

#include <cstring>

#include <unistd.h>

#include <netinet/in.h>

std::atomic<bool> running(true);

std::mutex connection\_mutex;

std::vector<std::thread> threads;

void signal\_handler(int signal) {

if (signal == SIGTERM || signal == SIGINT) {

running.store(false);

std::cout << "Received termination signal, shutting down..." << std::endl;

}

}

void handle\_client(int client\_socket) {

char buffer[1024];

ssize\_t bytes\_read;

while ((bytes\_read = read(client\_socket, buffer, sizeof(buffer) - 1)) > 0) {

buffer[bytes\_read] = '\0';

std::cout << "Received: " << buffer << std::endl;

// Echo back the received message

write(client\_socket, buffer, bytes\_read);

}

close(client\_socket);

}

int main() {

signal(SIGTERM, signal\_handler);

signal(SIGINT, signal\_handler);

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

if (server\_socket == -1) {

std::cerr << "Error creating socket: " << strerror(errno) << std::endl;

return 1;

}

sockaddr\_in server\_addr{};

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

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

server\_addr.sin\_port = htons(8080);

if (bind(server\_socket, (sockaddr\*)&server\_addr, sizeof(server\_addr)) == -1) {

std::cerr << "Error binding socket: " << strerror(errno) << std::endl;

close(server\_socket);

return 1;

}

if (listen(server\_socket, 10) == -1) {

std::cerr << "Error listening on socket: " << strerror(errno) << std::endl;

close(server\_socket);

return 1;

}

std::cout << "Server is listening on port 8080..." << std::endl;

while (running.load()) {

sockaddr\_in client\_addr{};

socklen\_t client\_addr\_len = sizeof(client\_addr);

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

if (client\_socket == -1) {

if (errno == EINTR) {

// Interrupted by signal, exit gracefully

break;

}

std::cerr << "Error accepting connection: " << strerror(errno) << std::endl;

continue;

}

std::lock\_guard<std::mutex> lock(connection\_mutex);

threads.emplace\_back(std::thread(handle\_client, client\_socket));

}

// Wait for all threads to finish

for (auto& thread : threads) {

if (thread.joinable()) {

thread.join();

}

}

close(server\_socket);

std::cout << "Server shutdown gracefully." << std::endl;

return 0;

}

Client:

#include <iostream>

#include <cstring>

#include <unistd.h>

#include <arpa/inet.h>

int main() {

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

if (client\_socket == -1) {

std::cerr << "Error creating socket: " << strerror(errno) << std::endl;

return 1;

}

sockaddr\_in server\_addr{};

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

server\_addr.sin\_port = htons(8080);

if (inet\_pton(AF\_INET, "127.0.0.1", &server\_addr.sin\_addr) <= 0) {

std::cerr << "Invalid address: " << strerror(errno) << std::endl;

close(client\_socket);

return 1;

}

if (connect(client\_socket, (sockaddr\*)&server\_addr, sizeof(server\_addr)) == -1) {

std::cerr << "Connection failed: " << strerror(errno) << std::endl;

close(client\_socket);

return 1;

}

std::string message;

char buffer[1024];

while (true) {

std::cout << "Enter message (or 'exit' to quit): ";

std::getline(std::cin, message);

if (message == "exit") {

break;

}

send(client\_socket, message.c\_str(), message.size(), 0);

ssize\_t bytes\_received = recv(client\_socket, buffer, sizeof(buffer) - 1, 0);

if (bytes\_received > 0) {

buffer[bytes\_received] = '\0';

std::cout << "Server response: " << buffer << std::endl;

}

}

close(client\_socket);

return 0;

}