MaheedharaSai.T
AVV,ECE

Project Documentation

LAN Chat Application

Project Summary

A compact LAN chat system implemented in C++ for Linux. Uses:
*TCP for reliable client-server chat
* UDP broadcasts for local discovery of servers

Files Provided

1. tcp_server.cpp (multithreaded TCP server)
2. tcp_client.cpp (console TCP client)
3. udp_discovery.cpp (UDP broadcast/listen utility)
4. Makefile (build helper)
5. README.md (quick start)
6. Common.h

Build & Run Instructions

1. On Linux with g++:
$ make
2. Start server:
$ ./tcp_server
3. Start client (from another machine on same LAN):
$ ./tcp_client <server-ip>
4. Optional discovery:
Server: ./udp_discovery listen
Client: ./udp_discovery bcast

Protocol & Message Format

This simple app transmits raw text lines over TCP, terminated by newline characters.
No length-prefix framing or binary protocol is used—suitable for demo/personal LAN use only.

Testing Plan

- Manual test: Start server, connect multiple clients, verify messages are relayed.
- Network test: Run on different machines/subnets, verify UDP discovery when broadcasting allowed.
- Failure test: Close client abruptly and ensure server cleans up sockets.

MaheedharaSai.T
AVV,ECE

Security Considerations

- No authentication or encryption. Do NOT run on untrusted networks.
- Consider TLS and authentication for sensitive use.
- Validate/limit message sizes to avoid resource exhaustion.

Extending this Project

1. Use non-blocking I/O + epoll or asio for scalability.
2. Add message framing (length prefix).
3. Add username handling and server-side logging.
4. Add UDP peer-to-peer chat and NAT traversal techniques.
5. Add GUI (Qt, GTK) or web interface via websocket bridge.

CODE

File: tcp_server.cpp

```
// tcp_server.cpp
// Simple multithreaded TCP chat server for LAN.
// Build: g++ -std=c++17 tcp_server.cpp -pthread -o tcp_server
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <unistd.h>
#include <atomic>
#include <cstring>
#include <iostream>
#include <mutex>
#include <set>
#include <string>
#include <thread>
#include <vector>
constexpr int SERVER_PORT = 9009;
constexpr int BACKLOG = 10;
constexpr int BUF_SIZE = 2048;
std::set<int> clients;
std::mutex clients_mtx;
std::atomic<bool> running{true};
void broadcast(const std::string &msg, int except_fd = -1) {
std::lock_guard<std::mutex> lock(clients_mtx);
for (int fd : clients) {
if (fd == except_fd) continue;
send(fd, msg.c_str(), msg.size(), 0);
}
}
```

MaheedharaSai.T
AVV,ECE

```
void handle_client(int client_fd) {
char buf[BUF_SIZE];
while (running) {
ssize_t n = recv(client_fd, buf, sizeof(buf) - 1, 0);
if (n <= 0) break;
buf[n] = '\0';
std::string msg = std::string(buf);
// Optionally add sender id prefix
broadcast(msg, client_fd);
}
close(client_fd);
{
std::lock_guard<std::mutex> lock(clients_mtx);
clients.erase(client_fd);
}
std::cerr << "Client disconnected: " << client_fd << "\n";
}
int main() {
int listen_fd = socket(AF_INET, SOCK_STREAM, 0);
if (listen_fd < 0) { perror("socket"); return 1; }
int opt = 1;
setsockopt(listen_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
sockaddr_in srv{};
srv.sin_family = AF_INET;
srv.sin_addr.s_addr = INADDR_ANY;
srv.sin_port = htons(SERVER_PORT);
if (bind(listen_fd, (sockaddr*)&srv, sizeof(srv)) < 0) { perror("bind"); return 1; }
if (listen(listen_fd, BACKLOG) < 0) { perror("listen"); return 1; }
std::cout << "TCP Server listening on port " << SERVER_PORT << "\n";
std::vector<std::thread> threads;
while (running) {
sockaddr_in cli{};
socklen_t cli_len = sizeof(cli);
int client_fd = accept(listen_fd, (sockaddr*)&cli, &cli_len);
if (client_fd < 0) {
perror("accept");
break;
}
{
std::lock_guard<std::mutex> lock(clients_mtx);
clients.insert(client_fd);
}
std::cout << "Client connected: " << client_fd << "\n";
```

```
threads.emplace_back(handle_client, client_fd);
}
running = false;
close(listen_fd);
for (auto &t : threads) if (t.joinable()) t.join();
return 0;
}
```