

CSC4220/6220: Computer Networks

Team Project

Introduction

In this project, you will build a basic chat server supporting multiple clients over the Internet. This project will allow you to learn about network protocols by implementing a simple version of a chat program inspired by Internet Relay Chat (IRC) principles. You can use any programming language you prefer.

Project Objectives

1. Design an Object-based Protocol

a. Chat Protocol

This simplified chat protocol uses text-based commands for easy implementation and testing. Here are the main commands your client should support:

Command	Description
/connect <server-name> [port#]	Connect to named server (port# optional)
/nick <nickname>	Pick a nickname (should be unique among active users)
/list	List channels and number of users
/join <channel>	Join a channel, all text typed is sent to all users on the channel
/leave [<channel>]	Leave the current (or named) channel
/quit	Leave chat and disconnect from server
/help	Print out help message

You may refer to the original IRC protocol described in [RFC 1459](#) for more insights into chat protocols.

b. Object-based Protocol

A client can accept the IRC commands but all communication with the server is using objects. Design an object-based protocol that gives the same functionality as the subset of IRC given earlier. Note that you will not only have objects that deliver commands, but you will also need objects that deliver events.

2. Create the Chat Server

The chat server, called *ChatServer*, should:

- Accept the following arguments: `-p <port#> -d <debug-level>`, where debug-level is 0 (errors only) or 1 (all events).
- Go idle if unused for over three minutes, automatically shutting down.
- Gracefully shut down with user command Ctrl-C.

It is recommended to develop the server in the following stages. Note that you won't necessarily write the code for each version but thinking in stages will make it easier to design the **multi-channel, multithreaded server** that we want to develop.

- **Single-Channel, Single-Threaded Server:** Start with a single-threaded, single-channel server.
- **Multi-Channel, Single-Threaded Server:** Expand to support multiple channels.
- **Multi-Channel, Multi-Threaded Server:** Add threads, with a maximum of **four** concurrent threads to avoid overload.

3. Create the Chat Client

Create a simple text-based chat client to connect to the server and demonstrate its core functionality. The client should be named *ChatClient*. It takes no command line arguments (as we can specify server name and port number via the connect command).

4. Extra Credit

- Use colored terminal fonts.
- Gracefully shut down with user command Ctrl-C.
- Implement basic logging for server messages and user activity.

Deliverables

- Demo video: A five-minute demo posted online (e.g., YouTube) and linked in your README.md file. Show the chat server and client in action, with a brief code walkthrough.
 - Tip: For recording the video, you can use Teams, Webex (the university has site licenses for both; access via your GSU account), or other licensed video recording tools.
- Project files: A zip file named “[Team#]_[TeamName]_[LastName1]_[LastName2]_[...etc].zip” including
 - All your source code
 - A read me file named “README.md” indicating:
 - Names of IDs of all team members,

- A web link to the demo video,
- A file/folder manifest to guide reading through the code,
- A section on building and running the server/clients,
- A section on how you tested it,
- A section on observations/reflection on your development process and the roles of each team member.

Grading Criteria

- Total: 100 points (+15 extra credits)
- Basic functionality: 70 points
- Multi-threaded: 20 points
- Demo video: 5 points
- Documentation: 5 points
- Extra credits
 - Colored fonts: +5 points
 - Shut down with Ctrl-C: +5 points
 - Logging: +5 points

--

This is a team project. You must not collaborate with anyone beyond your team.

Online resources must be properly referenced.

If you use GenAI, you MUST specify which part of your code is generated/modified by it. You MUST also submit a separate file of your interaction history including all your prompts and its response.