*Department of Cyber Defense, School of Cybersecurity, Korea University*

**CYDF 321 – Computer Networks (컴퓨터네트워크와실습)**
**Instructor: Prof. Wonjun Lee**

# Term Project, Spring 2022
**Due: June 14, 2022 11:59:59 PM (Midnight)**

## ◆ Term Project: Implementing a 1:N Multi-Threaded Chat Application using Socket API

### A. Objective

- To comprehend how the TCP specification is implemented in Berkeley Software Distribution (BSD) sockets (i.e., Berkeley sockets)
- To learn how to implement a multi-threaded network application using the socket and thread API.

### B. Overview

A socket is a software interface for application processes to send and receive messages through a computer network. It typically serves as an endpoint of TCP/IP protocols between two processes. As an abstract representation for a local endpoint, the socket API makes application developers not need to take care of the specifics of transport-layer implementations including congestion control, flow control, error recovery, and multiplexing for process-to-process communications. BSD socket programming API, proposed in the early stage of TCP/IP, is a 'de-facto' standard in C/C++ languages for UNIX/Windows environment. While its simple abstraction separates the network application developers from transport services, the socket has enabled the tremendous growth of network applications on the Internet, by lowering the barriers to entry.

The socket API is one of the most widely used POSIX APIs, a means of understanding the key concepts in Unix-like operating systems such as file descriptors and signals, and is an integral part of implementing a network application idiomatically. For the application developers, programming with socket APIs requires some experience and subtle details, resulting in the performance implication in some applications.

In this project, we implement a multi-threaded chat application based on the client-server architecture, where a server and multiple clients communicate with each other using TCP sockets.

### C. Components

i. Implementation Part

The implementation part consists of a *chat server* and multiple *chat clients*. We assume that the maximum number of clients is 5. First, a server creates a TCP socket, waiting for an incoming connection from clients. Then, a client establishes a TCP connection to the server. If the TCP connection is successfully established, the client's IP address and port number are printed on the server side. The success message is printed on the client side, confirming the connection state. Other clients can additionally establish TCP connections with the same server. Each client sends a message to the server, and the server prints the received message on server's screen. Then, the server sends the received message to all other connected clients, and the same message is printed on other client sides. The server only forwards the received message from each client to all other clients. In other words, there is no message made on the server side. The clients exchange messages through the server, and the exchanged messages are printed on server and other client sides. After this process, the sockets are closed.

ii. Description Part (Report)

The description must address how the application works (both the server and the clients), and the implementation details. It also includes an execution environment, how to build and execute your programs, plus the screen dumps of the executions on the server and clients.

## D. Requirements

i. Environment

You MUST write the programs using C programming language on POSIX-compliant operating systems (e.g., Linux, macOS, UNIX-like, …). The BSD socket API is used to make the programs communicate with each other. Specifically, your programs should include the following BSD socket functions: `socket`, `bind`, …, and `close`. In addition, the following functions in <`pthread.h`> can be used for multi-threading C programming: `pthread_create`, `pthread_detach`, `pthread_join`, `pthread_mutex_lock`, and `pthread_mutex_unlock`.

ii. Chat Server

The server MUST take one command-line argument as a port number for the program. The server only accepts up to 5 TCP clients for the chatting program. If the TCP connection is made with a client, the client's IP address and port number are printed to its standard output (e.g., "Connection from 163.152.162.144:12345"). After the connection, the server receives the nickname of the connected client (e.g., "User1", "User2"). Whenever a new client connects to the server, the server uses client's nickname to notify other connected clients of the new connection and also displays it on the server's screen (e.g., "User2 is connected").

If the server receives a message from a client, it should be printed on the screens of the server and other clients. On the screens of the server and other clients, the nickname of the client who sent the message should be printed together in front of the message (e.g., "User1: Hello"). On the server side, it does not receive a string from the prompt but only sends the received message from each client to other clients with the nickname of a client. At this time, the received message is not sent back to the client that sent the message. In this term project, the server communicates with multiple clients using multi-threading. The process of receiving and delivering messages over several sockets by the server should be done *simultaneously*, not sequentially. If there is a variable shared by multiple threads, you should set a critical section so that only one thread can access the variable at a time, and modify the variable only within the critical section. Otherwise, the result may be different depending on the order of threads accessing the variable.

If a client named User2 sends the message "QUIT" to the server, the server sends the message "User2 is disconnected" to other clients and also prints it on server's screen. Then, the server closes its socket communicating with the client named User2. It repeats this operation.

iii. Chat Client

The client MUST take three command-line arguments: an IP address of the server, a port number that the server designates for the application, and the nickname of the client. The client makes a TCP connection with the server using the command-line arguments (an IP address and a port number) in this application. After establishing the successful connection, the client sends its nickname to the server first by using the third command-line argument. A message meaning the successful connection state and its nickname is displayed on the screens of the server and all the clients including itself (e.g., "User1 is connected").

When a message sent by a client named User2 is printed on the screen of a client named User1, the

message should be printed with the nickname of the sending client (e.g., "User2: Hello"). However, if a client (i.e., User2) sends a message, it can output the message on its own screen without its nickname (e.g., "Hello"). The client need to use multi-threading for its simultaneous transmission and reception.

If a client named User2 sends the message "QUIT", the message "User2 is disconnected" is printed on the screens of a server and all the clients including User2. The client entering "QUIT" closes its socket and the client program is terminated.

iv. Report

Your report MUST contain the followings:

- Execution environment and how to build the programs
- A detailed explanation of the BSD socket API and its interactions with TCP in a multi-threaded application
- Comments for each line of the essential part of your code
- The screen dumps and analysis of execution results

## E. Evaluation & Deliverable

i. **No excuse (except the Blackboard system problem) will be accepted for late submissions!**

ii. When there is a suspicion that a submission of one student is similar to that of another student(s), those students have obligation to be interviewed for questions related to their work. You can refer to (but not copy and paste) any information from the Internet, but do not just copy the material of the other CYDF321 student(s). There is a demerit mark (severe minus point), if you copy the materials from other students'.

iii. **Submit your source code build script (if any) in proper file formats**. If your code is not built and executed properly, you can have a demerit mark.

iv. **Submit your project report in 'PDF format', NOT others** (e.g., hwp, pages). Note again that **you *should* check the requirements above to avoid extra demerit marks before submitting your report**.

v. How to submit your work

1. **Compress** all your materials including source code files, executables, and reports **in one file.**
2. The filename should be "`[CYDF321-TP] StudentID-Name.ext`". (e.g., `.ext` might be `.zip`, `.tar`, `.7z`, etc.)
3. Upload to Assignment submission menu of Blackboard.
4. **If there is a problem on Blackboard, send an e-mail attaching your file to chief TA Jieun Yu (jieunyu00@gmail.com).** Please let your e-mail 'subject' follow the form of "**[CYDF321-P1] Student ID-Name**" (e.g., [CYDF321-TP] 2020339999-홍길동).