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

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

# Project 1, Spring 2022
**Due: May 4, 2022 11:59:59 PM (Midnight)**

## ◆ Project 1: Implementing a 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 network application using the socket 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 simple chat application based on the client-server architecture, where a server and a client communicate with each other using TCP sockets.

### C. Components

i. Implementation Part

The implementation part consists of a *chat server* and a *chat client*. First, the server creates a TCP socket, waiting for an incoming connection from the client. Then, the 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. Then, the client and server start exchanging messages with each other, and the exchanged messages are printed on either side. After this process, the sockets are closed.

ii. Description Part (Report)

The description must address how the application works (both the server and the client), 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 client.

## 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`.

### ii. Chat Server

The server MUST take one command-line argument as a port number for the program. The server permits only one TCP client for the chatting program. If the TCP connection is made, the client's IP address and port number are printed to its standard output (e.g., "Connection from 163.152.162.144:12345"). The received messages from the client should be printed on the screen of the server. After then, the server sends the string input from the user to the client. Note that the server does not need to support multi-threading at this time. In other words, receiving and sending a string over its socket does not happen *simultaneously*, but sequentially. The chatting program continues to run until the user enters the message "QUIT". If the server sends (or receives) the message "QUIT", the message "Disconnected" is printed on the screen, the socket is closed and the program is terminated.

### iii. Chat Client

The client MUST take two command-line arguments: an IP address of the server, and a port number that the server designates for the application. The client makes a TCP connection with the server using the command-line arguments in this application. After establishing the successful connection, the message "Connected" is printed on the screen. The client sends a message to the server first by user's input. After then, the client receives a message from the server, and it should be printed to its standard output. It repeats this operation sequentially. Note that the client does not need to use multi-threading for its simultaneous transmission and reception. If the client sends (or receives) the message "QUIT", the message "Disconnected" is printed on the screen, the socket is closed, and the 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
- 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-P1] 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-P1] 2020339999-홍길동).