ECE 448/528
Application Software Design

Lecture 6. TCP Server Design I
Spring 2025

Won-Jae Yi, Ph.D.

Department of Electrical and Computer Engineering
Illinois Institute of Technology

Java Socket Programming

Socket Programming

- Most languages support low-level network programming via network sockets.
- Since sockets from different languages are for the same set of network communications, they look very similar.
- Let's learn Java socket programming.
  - You'll be able to learn socket programming for other languages by yourself
  - A good example showing how to learn a new library.
    - Step 1: get to know the area this library covers.
    - Step 2: use a tutorial to learn key classes and methods.
    - Step 3: learn more details from the documentation.
- Recall that we are building a TCP server to reverse strings.

Step 1: How TCP server works?

- The server starts by opening a port that is known to the clients.
- The server then waits for a client to connect.
- If a client is connected, the server may serve the client.
  - Receive data from the client via an input byte stream.
  - Send data to the client via an output byte stream.
- The server may concurrently serve multiple clients while waiting for new clients to connect.

Step 2: Key Classes and Methods

- Create a `ServerSocket` object to open a port.
- Use `ServerSocket.accept` to wait for a client.
- `ServerSocket.accept` returns a `Socket` object representing a connected client.
  - Obtain the input stream via `Socket.getInputStream`.
  - Obtain the output stream via `Socket.getOutputStream`.
- Create a `Thread` object for each client.
  - A Java program can execute multiple threads
  - The server can serve each client in a separate thread, and continue to wait for new clients at the same time.

Step 3: Additional Details

- Tools for troubleshooting.
  - Is the port open for sure?
  - What clients are actually connected?
- Obtain the address of a client as its identification.
  - For troubleshooting and security purposes.
- Use additional classes and methods to control more aspects of the TCP server.

Testing with a TCP Client

- Use a remote terminal tool to test our server.
  - Save time to implement our own.
  - e.g., telnet and PuTTY
  - Use telnet as it is the most convenient one for manual testing.
    - Note that it may or may not fully support UTF-8.
- That is one of the reasons we don't provide unit tests for our TCP server.
- Actual network communications are only tested during integration tests.
  - Need to set up the network environment, e.g., to make sure there is no other application using the same port.

Multithreaded TCP Server Design

ReverseServer

- `src/main/ece448/lec06/ReverseServer.java`
- Under branch `lec06-accept`.
- The constructor takes port as a parameter.
- Used to construct the ServerSocket member server.
- server is final as we should not change it later.
- We first see if a client can connect.
- Use a loop where each iteration starts using accept to wait for a client to connect.
- Display its address and then close it without any communication.
- Use logger to output useful informations so you'll know what's going on.

ReverseServer (Cont.)

- Errors are reported via `Exceptions`.
- `implements AutoCloseable` for resource management.
  - For resources like sockets that are not managed by GC, via the try-with-resources statement.
  - Override the `close` method to close the `server`.
  - Use `@Override` to indicate that `close` should override a method from an implemented interface or base class.
- The `main` method uses the try-with-resources statement to create a `ReverseServer` object and to close it when `main` exits, even when an exception is thrown.
  - Much more concise than using `finally`.
- Run the server.
  - Use the command `netstat -tan` to see TCP connections.
  - Use telnet to connect to the server and see how it reacts.

Multithreaded ReverseServer

- `src/main/ece448/lec06/ReverseServer.java`
  - Under branch `lec06-thread`.
- Create a new thread to serve each client.
  - via the class `ReverseProcessor`
  - Class design principle: each class should have its own responsibility – waiting for new clients and serving a client is different.
- Use `setDaemon(true)` to tell JVM that JVM could exit while `t` is still running.
- Use `start` to start the thread.
  - The main thread will start the next loop iteration.
  - The newly created/started thread will serve the client.

ReverseProcessor

- `src/main/ece448/lec06/ReverseProcessor.java`
  - Under branch `lec06-thread`.
- `implements Runnable` for multithreading
  - Override the `run` method, which is the entry point of the new thread.
- The constructor runs in the main thread.
  - Store the `client` object passed by `ReverseServer`.
  - The `client` object will be used by the `run` method later in the new thread.
  - This is a typical way to pass data from one thread to another.
- We don't reverse strings here; just log what is sent by `telnet`.

How telnet works?

- By default, telnet will wait until you press enter to send the whole line.
  - The enter key will be interpreted as two characters "\r\n“
  - e.g., 0x0D 0x0A.
- Can we use this feature to allow users to input multiple strings for reversing and to exit telnet gracefully?