ECE 448/528
Application Software Design

Lecture 7. TCP Server Design II
Spring 2025

Won-Jae Yi, Ph.D.

Department of Electrical and Computer Engineering
Illinois Institute of Technology

Implementing ReverseProcessor

Handling Multiple Strings

- Recall `telnet` separates lines by "\r\n".
- User story to support multiple strings:
  - As an end-user, I want to send multiple strings separated by "\r\n" to the server to be reversed via UTF-8 encoding, so that I can use telnet to test the server.
- User story to support end of request:
  - As an end-user, I want to send an empty string to the server to indicate I am done, so that the connection can be closed gracefully.
- Which one to implement first?
  - Take the simpler one!

Obtain String and Disconnect if Requested

- `src/main/ece448/lec06/ReverseProcessor.java`
  - Under branch `lec07-message`.
- Use ByteArrayOutputStream to buffer bytes before they are converted to a String
- Ignore \r since that's not part of the string.
- When \n arrives, extract the string via toString.
  - Decode from UTF-8.
  - Exit if the string is empty.
  - Log and continue to the next string.
  - Don't forget to reset the buffer!

Reverse and Send Back

- `src/main/ece448/lec06/ReverseProcessor.java`
  - Under branch `lec07-reverse`.
- Call ReverseString.reverse to reverse the string.
- Call getBytes("UTF-8") to get the bytes.
- Send the bytes back using OutputStream.write.
- Send "\r\n" so that telnet will display the reversed string in its line.

Discussions

General TCP Server Architecture

- Create a server socket to listen on a port, then use a loop to wait/accept new clients.
- Once a client arrives, send the client socket to a separated thread to serve the client. Then, the server socket may continue accepting additional clients.
- To process requests from a client, use another loop to read from the input byte stream.
  - When a complete message is received, the client will perform some computation to obtain a result.
  - The result will be sent back to the client via the output byte stream.
- Need a presentation layer protocol to decide
  - How to extract messages from the input byte stream.
  - How to convert results to the output byte stream.

Design Considerations

- The two responsibilities of waiting for clients and serving a client can be handled by two classes.
- If a different kind of TCP service should be implemented, the class that waits for clients to connect will mostly remain the same.
- However, we may need to rewrite the class that serves a client.
- “Sandwich” structure to process requests from clients
  - Decode a request message from the input TCP stream.
  - Do some computations to obtain a response.
  - Encode and send the response via the output TCP stream.
- A well-designed framework may allow us to focus on the application layer only.
  - To only consider how to compute responses from requests, without worrying about how they are decoded/encoded or sent/received over the network.

Performance Considerations

- It is not efficient to read one byte at a time from the input TCP stream.
  - Prefer to read all available bytes at a time – but this requires additional code to handle the buffer properly.
- It is “expensive” to create and maintain a thread for each client.
  - A major problem when you need to serve thousands or more clients at the same time.
  - OS needs to allocate memory for each thread.
  - Frequent context switches consume a lot of processor power.
  - Prefer to use thread pools or I/O multiplexing techniques that use less threads – but this requires substantial change to the whole design.
- We would move to a well-designed framework for our projects to meet the performance demands you may face in the future.

General TCP Server Architecture

- Create a server socket to listen on a port, and then use a loop to wait/accept new clients.
- Once a client arrives, send the client socket to a separated thread to serve the client. Then the server socket may continue accepting additional clients.
- To process requests from a client, use another loop to read from the input byte stream.
  - When a complete message is received, the client will perform some computation to obtain a result.
  - The result will be sent back to the client via the output byte stream.
- Need a presentation layer protocol to decide
  - How to extract messages from the input byte stream.
  - How to convert results to the output byte stream.

Design Considerations

- The two responsibilities of waiting for clients and serving a client can be handled by two classes.
- If a different kind of TCP service should be implemented, the class that waits for clients to connect will mostly remain the same.
- However, we may need to rewrite the class that serves a client.
- "Sandwich" structure to process requests from clients
  - Decode a request message from the input TCP stream.
  - Do some computation to obtain a response.
  - Encode and send the response via the output TCP stream.
- A well-designed framework may allow us to focus on the application layer only.
  - To only consider how to compute responses from requests, without worrying about how they are decoded/encoded or sent/received over the network.

Performance Considerations

- It is not efficient to read one byte at a time from the input TCP stream.
  - Prefer to read all available bytes at a time – but this requires additional code to handle the buffer properly.
- It is costly to create and maintain a thread for each client.
  - A major problem when you need to serve thousands or more clients at the same time.
  - OS need to allocate memory for each thread.
  - Frequent context switches consume a lot of processor power.
  - Prefer to use thread pools or I/O multiplexing techniques that use less threads – but this requires substantial change to the whole design.
- We would move to a well-designed framework for our projects to meet the performance demands you may face in future.