https://github.com/Halloum97/ser321-summer25-C-jhallou1

1. Command Line Tasks (17 points)

Using Linux

1. **Create a directory named `cli_assignment`.**
    - `mkdir cli_assignment`
2. **Change the current working directory to the new directory `cli_assignment`.**
    - `cd cli_assignment`
3. **Create a new file named `stuff.txt` using the `touch` command.**
    - `touch stuff.txt`
4. **Add multiple lines of text to the file using the `cat` command.**

```
cat > stuff.txt
Line 1 text
Line 2 text
Ctrl+D (to save and exit)
```

5. **Count the number of words and lines in the file `stuff.txt`.**
    - `wc stuff.txt`
        - This will display the number of lines, words, and characters in the file.
6. **Append more text to the file `stuff.txt`.**

```
cat >> stuff.txt
New line appended
Ctrl+D (to save and exit)
```

7. **Create a new directory `draft` in the current directory.**
    - `mkdir draft`
8. **Move the file `stuff.txt` to the `draft` directory.**
    - `mv stuff.txt draft/`
9. **Change your working directory to `draft` and create a hidden file `secret.txt`.**

```
cd draft
```

```
touch .secret.txt
```

10. **Create a copy of the `draft` directory and name it `final`.**
    - `cp -r draft final`
11. **Rename the `draft` directory to `draft.remove`.**
    - `mv draft draft.remove`
12. **Move the `draft.remove` directory inside the `final` directory.**
    - `mv draft.remove final/`
13. **List all files and subdirectories along with their permissions in the `cli_assignment` directory.**
    - `ls -la`
14. **List the contents of `NASA_access_log_Aug95.gz` without extracting it.**
    - `zcat NASA_access_log_Aug95.gz | head`
15. **Extract the file `NASA_access_log_Aug95.gz`.**
    - `gunzip NASA_access_log_Aug95.gz`
16. **Rename the extracted file to `logs.txt`.**
    - `mv NASA_access_log_Aug95 logs.txt`
17. **Move the file `logs.txt` to the `cli_assignment` directory.**
    - `mv logs.txt cli_assignment/`
18. **Read the top 100 lines of the file `logs.txt`.**
    - `head -n 100 logs.txt`
19. **Create a new file `logs_top_100.txt` containing the top 100 lines using I/O redirection.**
    - `head -n 100 logs.txt > logs_top_100.txt`
20. **Read the bottom 100 lines of the file `logs.txt`.**
    - `tail -n 100 logs.txt`
21. **Create a new file `logs_bottom_100.txt` containing the bottom 100 lines using I/O redirection.**
    - `tail -n 100 logs.txt > logs_bottom_100.txt`
22. **Concatenate `logs_top_100.txt` and `logs_bottom_100.txt` into `logs_snapshot.txt`.**
    - `cat logs_top_100.txt logs_bottom_100.txt > logs_snapshot.txt`
23. **Append the line `asurite: This is a great assignment` and the current date to `logs_snapshot.txt`.**

```
echo "jhallou1: This is a great assignment $(date)" >>
logs_snapshot.txt
```

24. **Read the `logs.txt` file using the `less` command.**
    - `less logs.txt`
25. **Print the `student_names` column from `marks.csv` without the header using `cut`.**

```
cut -d '%' -f 1 marks.csv | tail -n +2
```

26. **Print a sorted list of marks in `subject_3`.**

```
cut -d '%' -f 4 marks.csv | tail -n +2 | sort -n
```

    -
27. **Print the average marks for `subject_2` using `awk`.**

```
awk -F '%' '{sum+=$3} END {print sum/NR}' marks.csv
```

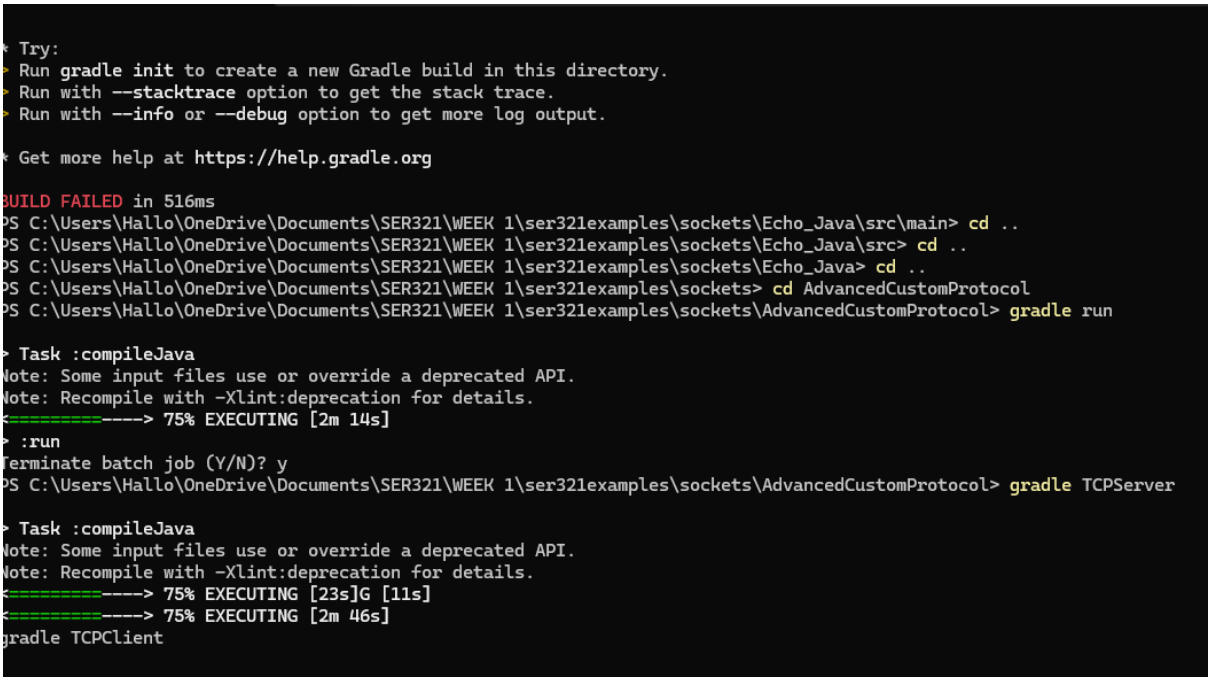
28. **Save the average into a new file `done.txt` inside the `cli_assignment` directory.**

```
awk -F '%' '{sum+=$3} END {print sum/NR}' marks.csv > done.txt
```

29. **Move `done.txt` into the `final` directory.**
    - `mv done.txt final/`
30. **Rename the `done.txt` file to `average.txt` inside the `final` directory.**
    - `mv done.txt average.txt`

2.2 Running examples

1. AdvancedCustomProtocol



This TCP client-server application allows the client to request jokes, quotes, or images from the server via JSON messages. The client sends a request for a specific type of content (joke, quote, image, or random) and receives a corresponding JSON response from the server. The server processes the request, sending back either a text-based joke or quote, or a Base64-encoded image, which the client can display in a graphical window. The application demonstrates basic TCP communication, handling JSON data, and serving different content types.

Gradle is involved in this project as a build automation tool that manages dependencies, compiles the Java source code, and runs the TCP/UDP client and server applications. The build.gradle file specifies the necessary dependencies, such as the org.json library for JSON handling, and defines custom tasks for running both the TCP and UDP versions of the client and server. These tasks (TCPServer , TCPClient , UDPServer , UDPClient) use Gradle's JavaExec to run the appropriate main classes, making it easy to execute the desired program without manually compiling and running the Java files.

2. JSON

```
PS C:\Users\Hallo\OneDrive\Documents\SER321\WEEK 1\ser321examples\Network\JSON> gradle JSON

> Task :JSON
ASU
Poly
[{"firstName":"John","lastName":"Doe"},{"firstName":"Anna","lastName":"Smith"},{"firstName":"Peter","lastName":"Jones"}]
John
Anna
Peter

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own sc
ripts or plugins.

See https://docs.gradle.org/7.4.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 647ms
2 actionable tasks: 1 executed, 1 up-to-date
PS C:\Users\Hallo\OneDrive\Documents\SER321\WEEK 1\ser321examples\Network\JSON>
```

This Java code demonstrates how to parse and manipulate JSON data using the org.json library. It starts by creating a JSON object (newObject) from a string that contains information about an organization, its address, and a list of employees. The program retrieves specific data from this JSON object, such as the organization's name and address. It then iterates through the employee list, extracts their first names, and creates a new JSON array (justFirstnames) containing only those first names. Finally, the program writes this new JSON array to a file called names.json.

Gradle is involved by managing the dependency on the org.json library, which is necessary for handling JSON objects and arrays. The build.gradle file specifies this dependency and uses a custom task to compile and execute the Java code, making it easier to manage dependencies and run the application seamlessly.

3.

```
PS C:\Users\Hallo\OneDrive\Documents\SER321\WEEK 1\ser321examples\Serialization\UserXml> gradle run

> Task :run
Ready to export a user
Done exporting a user as xml to user.xml
Importing a user as xml from user.xml
Read user: I AM

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own sc
ripts or plugins.

See https://docs.gradle.org/7.4.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 726ms
2 actionable tasks: 2 executed
PS C:\Users\Hallo\OneDrive\Documents\SER321\WEEK 1\ser321examples\Serialization\UserXml>
```

This code demonstrates XML serialization and deserialization using Java Beans and the XMLEncoder / XMLDecoder classes. The User class defines a serializable Java object with a parameterless constructor and getter/setter methods for each instance variable. The UserXMLSerialize class creates a User object and writes it to an XML file (user.xml) using XMLEncoder . It then reads the XML file back into a new User object using XMLDecoder , allowing the program to store and retrieve Java objects in XML format.

Gradle is involved through the application plugin, which defines the main class (UserXMLSerialize) and handles the build and execution of the project. Gradle automates

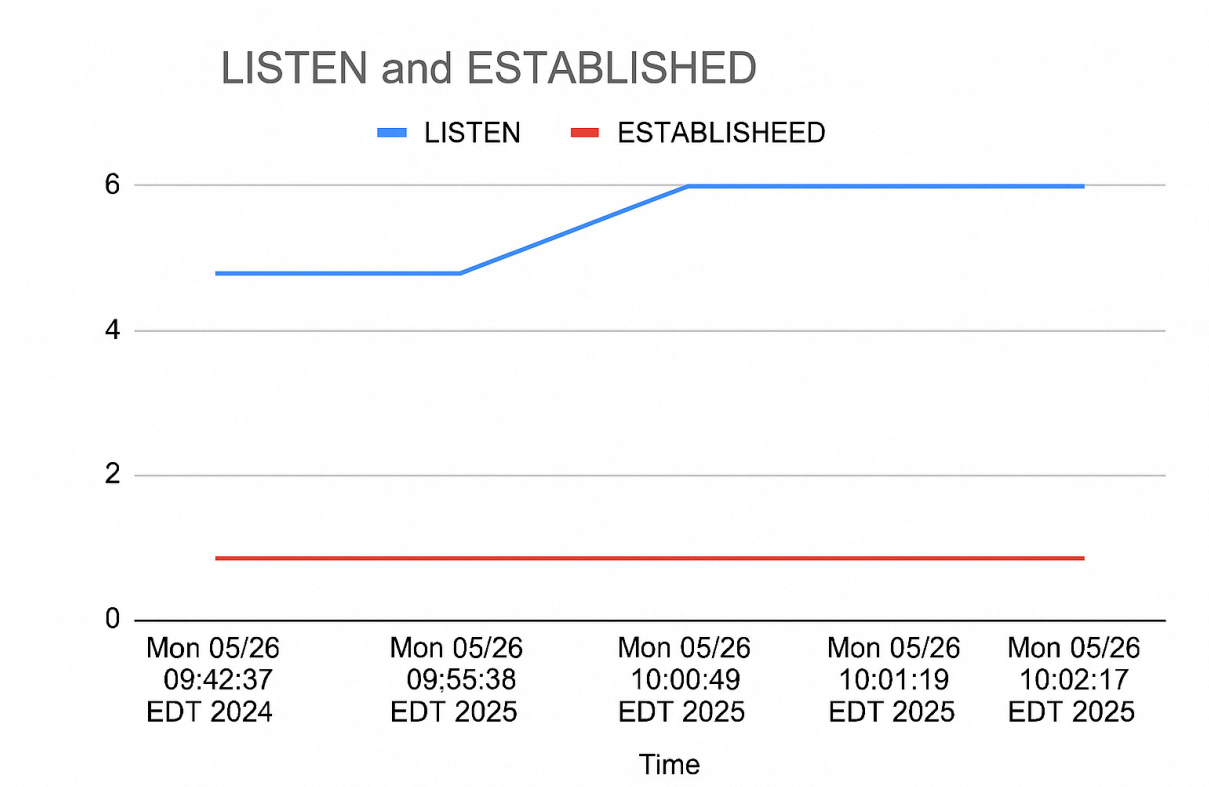
the process of compiling the code and running the application, ensuring all dependencies are correctly managed and the XML serialization example is executed seamlessly.

2.4
Screen cast link

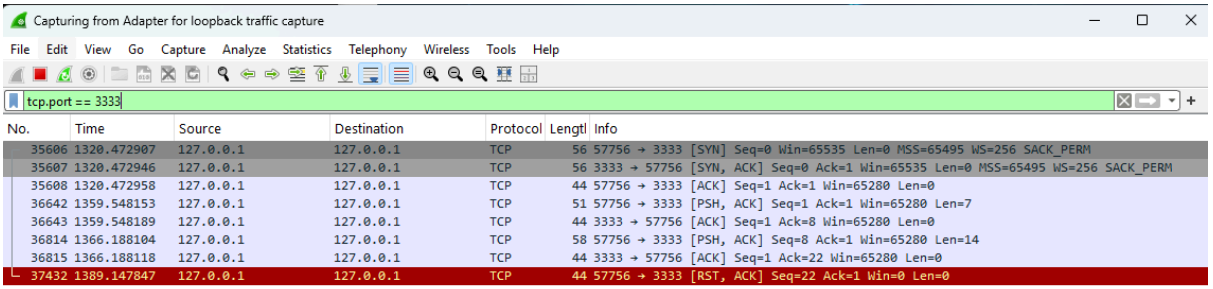
https://somup.com/cZ60fzH5NZ

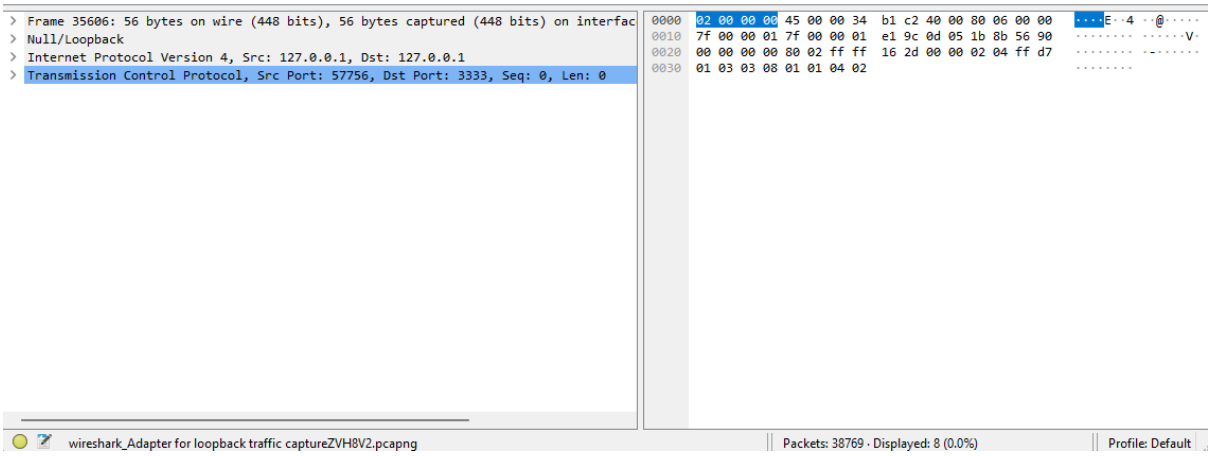
3.1

```
 while true; do
  echo "Time: $(date)"
  netstat -t | grep -E "ESTABLISHED|LISTEN"
  sleep 30
done
```



3.2





5a) Explanation of commands:
- ncat -k -l 3333: Starts a persistent TCP listener on port 3333.
- ncat 127.0.0.1 3333: Connects to the listener on the same port and sends the specified data.

5b) Packets for sending two lines:
- 7 packets were exchanged (3 for connection setup, 2 for data transfer, 2 for acknowledgment).

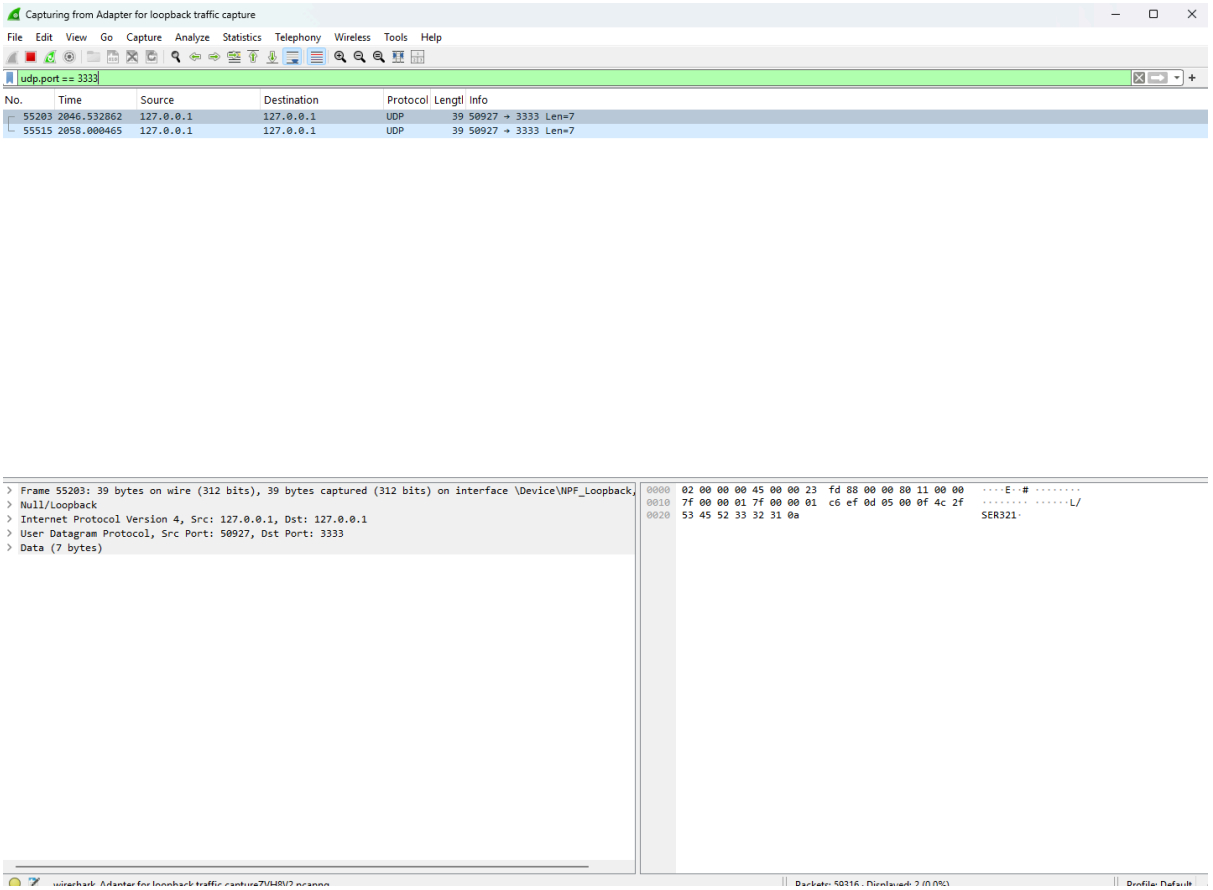
5c) Packets for the whole process:
- 9 packets total (including connection setup, data transfer, and connection teardown).

5d) Data bytes sent from client to server:
- 18 bytes (6 bytes for "SER321" and 12 bytes for "is amazing!!!").

5e) Total bytes over the wire:
- 378 bytes (including TCP/IP headers and the actual data).

5f) Overhead:

- 360 bytes of overhead (headers).
- 95.24% overhead, with only 4.76% actual data.



4a) Explanation of commands:
- ncat -k -l -u 3333: Starts a persistent UDP listener on port 3333.
- ncat -u 127.0.0.1 3333: Connects to the UDP listener on port 3333 and sends data.

4b) Packets for sending two lines:
- 2 UDP packets.

4c) Packets for the whole process:
- 2 UDP packets total (no connection setup/teardown in UDP).

4d) Data bytes sent:
- 12 bytes (6 bytes for "SER321" and 6 bytes for "Rocks!").

4e) Total bytes over the wire:
- 78 bytes (39 bytes per packet × 2 packets).

4f) Overhead:

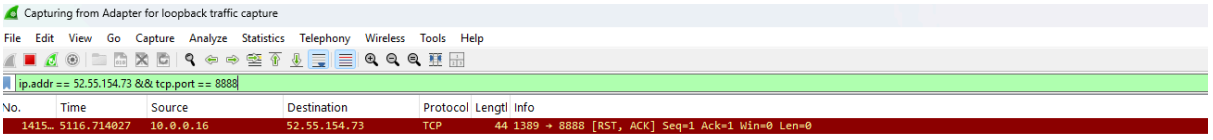
- 66 bytes of overhead (78 bytes total - 12 bytes data).

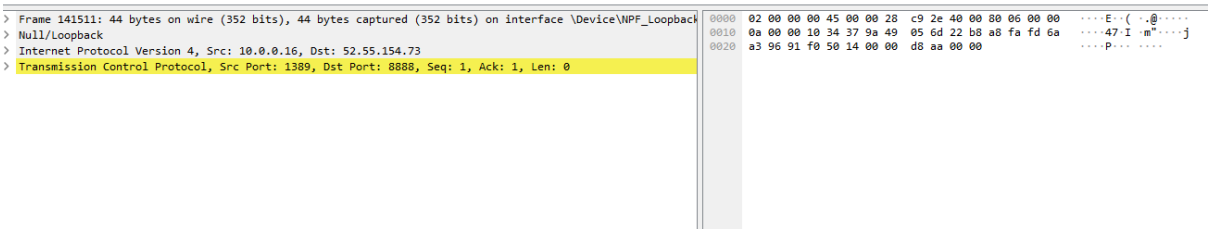
4g) Difference in relative overhead between UDP and TCP:
- UDP has less overhead because it doesn't establish a connection (no handshake or termination).
- TCP includes connection setup (SYN, ACK), data acknowledgment, and termination, leading to more overhead.

3.3.1

https://somup.com/cZ60oCHGXo

3.3.2





https://somup.com/cZ60D6HG2F

3.3.3
Does this work without issues?

- No, it will likely face challenges because of firewall rules, NAT, and local network restrictions.

Can you do it the same way as in 3.3.2?

- No, it's not as straightforward. AWS instances have public IPs and open ports (when configured correctly), while home networks often don't allow incoming connections easily.

What is different?

- The local server is behind a NAT and firewall, which blocks external connections. AWS has a public IP that's easily accessible, while your local machine's IP might not be.

3.3.4

Why can you easily reach your AWS server from a client in your local network but not the other way around?

- AWS servers have public IPs that are accessible to external networks. Your local server, however, is behind a NAT and firewall, which blocks incoming connections.

What can you do to reach your server in your local network from outside?

- You can set up **port forwarding** or use a **VPN** to allow external clients to access your local server.

What is the issue if you want to run your server locally and access it from the outside world?

- The main issue is that your local server is behind a NAT and firewall, making it difficult to reach from external networks without configuring port forwarding or setting up a VPN.