# Log Server Project Report

Student: Prestigiovanni William Teacher: Francesco Pedullà

ID: 2155657

## Introduction

This project is simply a "log server" – a server that logs data received from clients into a log file. The client and server executables are designed to be configurable through command-line options, with default values stored in a configuration file for flexibility.

## Specifications

1. Connection should be stateful: the client connects to the server, sends a set of messages and closes the connection.
2. The server should be able to manage an unlimited number of clients, with minimal delay to establish a connection.
3. Messages in the log file should include a header by the log server with timestamp and info on the client.
4. Log file is unique for all clients, but messages from different clients cannot overwrite or intertwine.
5. The client executable should accept the following options on the command line (default values should be stored in a configuration file):
   - IP address.
   - Listening port.
6. When started, the client should automatically connect to the server.
7. The client should wait for a new message from its standard input and send it immediately to the server.
8. The client should loop forever (i.e., till it reads an EOF).
9. When started, the server should open a new log file (without removing any old file).
10. The server executable should wait for input on a well-known port.
11. The server should shutdown when it receives a user-selected signal or a quit command from the command line.
12. The server executable should accept the following options on the command line (default values should be stored in a configuration file):
    - Listening port.
    - Directory to store the log file.

## Extra bonus

1. When the log file size exceeds a given threshold, the server should cancel the oldest log file in the log directory and create a new log file. In this case, the server should not create a new log file at start-up but rather append to the most recent log file in the directory.

2. Provide automated tests.

## Design

The architecture used is a simple `client-server architecture`. It is a widely used computing model that facilitates the distribution of tasks and responsibilities between two distinct entities: the client and the server. In this architectural paradigm, these entities interact to achieve a collaborative and efficient system.

### Architecture footprint

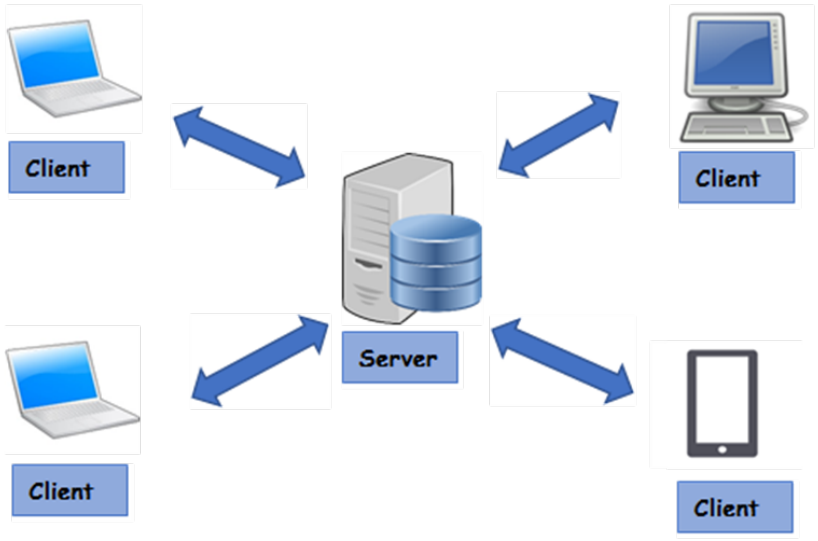


Figure 1: Example Architecture

The most important implementation choices are:

1. The protocol used is TCP (Transmission Control Protocol) that is a reliable and connection-oriented communication protocol within the Internet Protocol (IP) suite. It ensures the secure and orderly exchange of data between two devices by establishing a persistent connection, managing data flow and providing error detection and correction mechanisms.

2. The paradigm used to manage the client on the server is known as the "forking server" model. In this approach, upon receiving a client connection, the server creates a new dedicated child process to manage that specific connection concurrently.

3. The semaphore is used as a synchronization mechanism between processes to alternate writing to log files.

# Implementation

Addressing the following key challenges proved to be the most demanding aspects of the project:

1. Log file rotation: managing seamless log file rotation posed a significant challenge, requiring careful implementation to ensure the efficient handling of log data without disruptions.
2. Mutually exclusive access to log file: enforcing exclusive access to the log file presented a complex task. The implementation involved incorporating mechanisms to prevent conflicts when multiple processes attempted to write to the log simultaneously.
3. Graceful shutdown on SIGINT signal: ensuring a robust and orderly shutdown process in response to the SIGINT signal was crucial. Implementing proper cleanup procedures and resource closure was challenging to guarantee the correct termination of all components when the signal was received.

# Client-Server command-line options

As was said in the specification, both client and server have command line options. It will be shown these options below.

## Client

`./client localhost 9000` or `./client 127.0.0.1 9000`

By this configuration client will connect to **localhost:9000** or **127.0.0.1:9000**. The hostname is resolved by the client.

## Server

`./server 9000 logfile`

By this configuration, all log files will be saved in **current_working_dir/logfile** (current working directory where the file of project are in) and server will listen on port **9000**.

# Default configuration file

The client and server have default configuration files that are used when the user doesn't write anything as command-line options. In the configuration file of client there are the default IP address and listening port. For the server there are the default listening port and log file directory.

## Client

The client configuration file is stored at `config/config_client` and this is the content.

```
ADDRESS localhost
PORT 8080
```

## Server

The server configuration file is stored at `config/config_server` and this is the content.

```
PORT 8080
LOGPATH ../../log
```

# Shutdown Mechanism

## Server

The server is designed to shut down when it receives a user-selected signal from the command line. Pressing CTRL+C will gracefully terminate the server by SIGINT signal handled correctly.

## Client

The client is programmed to shut down gracefully when it receives a user-selected signal, EOF (CTRL+D) or CTRL+C from the command line. In both cases, the client correctly terminate because the two cases are handled correctly.

# Test

The tests to be done are the following:

1. Server startup (start the server only).
2. Client connect (after (1), record the connection on the log file).
3. Client message (after (1), let the client send a message and the server successfully logs it).
4. Second client message (after (3), start another client that sends a new message to the server).
5. Client shutdown (after (4), let the client successfully close the connection and the server logs the closing).
6. Server shutdown (after (5), let the server successfully stop, after recording the order of shutdown).

The **autorun.sh** script only compile and execute client and server. Basically it boots the server, a client 1 and a client 2. The other test cases MUST be run by the user.

To start the **autorun.sh**, use this command:

```
./autorun.sh
```

# Conclusion

In conclusion, this project successfully implemented a robust **client-server architecture** for logging data efficiently. Key accomplishments include the implementation of stateful connections, handling an unlimited number of clients and ensuring log file integrity. The introduction of log file rotation and exclusive access mechanisms demonstrated the commitment to scalability and data integrity.
In summary, this project not only met the specified requirements but also addressed additional challenges, making it a valuable contribution to efficient and secure log data management.

# Appendix

The project can be viewed on GitHub at this link `https://github.com/Prestijhonny/CSaP_project/tree/main`.