

# TW-Mailer Basic: Architecture and Implementation Report

---

## Authors

- **Client Implementation:** Tarik Yilmaz
  - **Server Implementation:** Peyman Aparviz
- 

## Architecture

The TW-Mailer Basic system is designed as a **Client-Server application** using a simple text-based protocol over TCP.

- **Server:** Implements an **Iterative Server** model. It handles one client connection at a time using strict blocking I/O. It accepts a connection, processes all commands ( `SEND` , `LIST` , `READ` , `DEL` ) from that client in a loop until the client disconnects or sends `QUIT` , and then moves to the next waiting connection. This simplifies concurrency management by avoiding race conditions on the file system within a single process.
- **Client:** A command-line interface (CLI) that connects to the server and allows the user to interactively send commands. It manages user input parsing and protocol formatting.

## Used Technologies

- **Language:** C++17.
- **Networking:** BSD/POSIX Sockets ( `<sys/socket.h>` , `<netinet/in.h>` , `<arpa/inet.h>` ) for reliable TCP communication.
- **File System:** strict usage of `std::filesystem` (C++17) for portable and robust file and directory manipulation (creating spool directories, listing files, removing messages).
- **Build System:** GNU Make.

## Development Strategy

The development followed an agile, iterative approach:

1. **Shared Foundation:** A `common.hpp` header was created first to standardize constants (buffer sizes, command strings) and shared utility functions ( `die` , `send_line` , `read_line` ). This ensures protocol consistency between client and server.
2. **Client Implementation:** The client was implemented to validate the protocol format, user input handling, and strict socket communication.
3. **Server Implementation:** The server was built to handle the defined protocol, with a focus on robust file persistence.
4. **Refinement:** Enhancements like the "Auto-List" feature and timestamp display were added on top of the working core.

## Implementation Details

### Persistence Strategy: Timestamps and Sequence Numbers

To store messages safely and order them correctly without a complex database, we use a structured filename format:

`YYYYMMDD_HHMMSS_SEQ.txt` (e.g., `20251215_223005_001.txt` ).

- **Why?** This format makes the filename **self-sorting**. An alphanumeric sort of the filenames typically corresponds to the chronological arrival order.
- **Sequence Number Check (Optimistic Write):** To determine the next available filename *without* scanning and counting all existing files (which would be slow  $O(N)$ ), we use an optimistic check:
  1. Generate the base timestamp (e.g., `..._223005` ).
  2. Start with sequence `001` .
  3. Check `if (fs::exists(filename_001))` .
  4. If it exists, increment (try `002` ) and repeat. If not, this is our file.
  - *Benefit:* In the vast majority of cases, `001` is available. We only do multiple checks if high traffic occurs within the exact same second. This avoids the cost of iterating the entire directory for every write.

### Reading and Ordering

While writing is optimized to avoid scanning, `LIST` , `READ` , and `DEL` operations require a consistent view of the mailbox.

- Since `fs::directory_iterator` does not guarantee any specific order, we iterate the directory, load all valid entries into a `std::vector` , and explicitly `std::sort` them.
- This ensures that "Message #1" is always the oldest message and "Message #N" is the newest, maintaining consistency between `LIST` indices and `READ` / `DEL` targets.

## Date and Time Display

The `LIST` command enhances the user experience by extracting the date and time directly from the filename.

- The server parses the `YYYYMMDD_HHMMSS` portion of the filename.
- It formats this into a readable string (e.g., `[2025-12-15 22:30:05]` ) and appends it to the subject line sent to the client.
- This allows the client to display timestamp information without needing to change the protocol structure (it simply prints the full subject line received).

## Auto-List Feature

To improve usability and prevent errors, the Client tracks the "state" of the user's view.

- **Logic:** The client maintains a `last_list_user` variable.
- **Behavior:** When a user attempts to `READ` or `DEL` a message, the client checks if the current username matches `last_list_user`.
- **Automation:** If they differ (or if `LIST` was never called), the client automatically triggers a `LIST` command *before* sending the `READ` or `DEL` command. This ensures the user (and the logical index they provide) is synchronized with the latest server state for that specific mailbox.