C++ Take-Home Coding Challenge: Stealth Service Activation Daemon

Objective: This exercise is designed to evaluate your ability to understand system requirements, make design decisions, and implement a robust C++ application. We are interested in your approach to problem-solving, concurrency, state management, and error handling.

Expected Time Commitment: We anticipate this assignment will take approximately 2-3 hours to complete. Please submit your solution at least the night before your scheduled technical interview.

Problem Overview:

You are tasked with creating a "Stealth Service Activation Daemon" (SSAD) in C++. The SSAD listens for a specific sequence of TCP connection attempts ("knocks") from client applications across a series of predefined ports. If the correct sequence of knocks is received from a unique source IP address within specified time constraints, the daemon "activates" a conceptual service for that IP address by logging an activation message.

This mechanism can be used to protect a service by not exposing its primary listening port until a client demonstrates pre-shared knowledge of the secret knocking sequence.

Core Requirements:

1. **Configuration:**
    - The daemon must be configurable via a simple text-based file (e.g., INI, JSON, YAML, or a custom format – your choice).
    - The configuration file must specify:
        - A sequence of 3 to 5 TCP trigger ports (e.g., `trigger_ports = 10001, 10002, 10003`).
        - An overall timeout for the *entire* sequence (e.g., `sequence_timeout_ms = 5000`). This is the maximum time allowed from the first valid knock to the last valid knock in the sequence.
        - An inter-knock timeout (e.g., `inter_knock_timeout_ms = 1000`). This is the maximum time allowed between two consecutive valid knocks in the sequence.
        - The path to an activation log file (e.g., `activation_log_file = /var/log/ssad_activations.log`).

2. **Daemon Behavior:**
    - Upon starting, the SSAD should parse its configuration. If the configuration is invalid, it should log an error and exit.
    - The daemon must listen for incoming TCP connection attempts on *all* configured trigger ports simultaneously.
    - **Knock Detection**: When a client attempts to connect to one of the trigger ports, the daemon should `accept` the connection and then immediately `close` it. The act of a successful `accept` (followed by an immediate `close`) on a trigger port constitutes a "knock." The daemon must record the source IP address of the knock and the port that was knocked.
    - **Sequence Tracking**:
        - The daemon must track the progress of the knocking sequence independently for each source IP address.
        - Knocks must arrive in the exact order specified in the configuration for the service to “activate”.
        - All knocks in a sequence must originate from the same source IP address.
        - The timing constraints (`sequence_timeout_ms` and `inter_knock_timeout_ms`) must be respected.
    - **Service Activation**:
        - If a source IP successfully completes the entire knocking sequence in the correct order and within all time constraints, the daemon must write an entry to the `activation_log_file`. The log entry should be in the format: `[YYYY-MM-DD HH:MM:SS] Service activated for IP: <source_ip>`
        - After successful activation (or any failure in the sequence), the state for that IP address should be reset, allowing it to attempt a new sequence.
    - **State Reset**: If a knock arrives out of order, or if any timeout is exceeded during an IP's attempt, its current knocking sequence progress must be reset. The IP must start the sequence from the beginning.
    - **Concurrency**: The daemon must be able to handle connection attempts and track sequences from multiple IP addresses concurrently and correctly.

3. **Technical Guidelines:**
    - **Language**: C++ (please specify the standard used, e.g., C++17, C++20).
    - **Libraries**: You may use standard C++ libraries. If you wish to use other libraries (e.g., Boost.Asio for networking), please note this choice and your reasoning.
    - **Error Handling**: Implement robust error handling (e.g., for socket operations, file I/O, configuration parsing). Log errors appropriately (to `stdout`/`stderr` or a dedicated daemon log file – your choice).

    - **Resource Management**: Ensure proper management of resources (e.g., file descriptors, memory).

Deliverables:

1. We prefer you share a github repo if it is doable in reasonable time. Otherwise, make a .zip or .tar available to us somehow.
2. **Source Code**: All C++ source files (`.h`, `.cpp`) and a `CMakeLists.txt` file.
3. **README.md**: A brief document explaining:
    - The format of your configuration file and an example.
    - Instructions on how to build and run the daemon.
    - Any assumptions you made.
    - (Optional) Known limitations or potential areas for future improvement. Be prepared to discuss these in person.

Evaluation Criteria:

Your submission will be evaluated on:

- **Correctness**: Does the daemon meet the specified requirements?
- **Design**: Clarity, modularity, and scalability of your design. How well is state managed? How is concurrency handled?
- **Code Quality**: Readability, maintainability, and use of C++ best practices.
- **Error Handling & Robustness**: How well does the application handle potential errors and edge cases?
- **Documentation**: Clarity of your `README.md` and comments where necessary.

Clarifications & Assumptions:

While we've tried to be clear, you may encounter ambiguities. **Feel free to email us questions.**

Please make reasonable assumptions, document them in your `README.md`, and be prepared to discuss them. The "service activation" (logging) is intentionally simple for this exercise. The focus is on the networking, state management, and concurrent processing aspects of the daemon.

We look forward to reviewing your solution and discussing it with you!