BlueAlert OS: Module Explanation for Bluetooth-Based Alert and File Sharing System

1 1. Project Overview

BlueAlert OS is a lightweight, decentralized, Bluetooth-based system designed for broadcasting emergency alerts and sharing files without internet connectivity. It operates as a userspace daemon on Linux systems, leveraging Bluetooth for peer-to-peer communication. The system is modular, beginner-friendly, and scalable through multi-hop message relaying, making it suitable for scenarios like disaster response or offline file sharing.

This document outlines the system's modules, their roles in the front-end (user interaction) and back-end (core functionality), and how they work together to achieve the project's goals.

2 2. System Architecture

BlueAlert OS runs on Bluetooth-enabled devices (e.g., Linux laptops) and uses a peer-to-peer model where devices broadcast, receive, and relay messages or file chunks. The front-end provides minimal user interaction (e.g., sending alerts or files), while the back-end handles Bluetooth communication, message management, and file processing.

Key features include:

- Broadcasting and receiving alerts via Bluetooth.
- Multi-hop relaying to extend range.
- File splitting, transfer, and reassembly.
- Local storage to prevent duplicate broadcasts.

3 3. Modules and Their Roles

Below are the five core modules of BlueAlert OS, implemented in C using the BlueZ Bluetooth API or socket-based simulation. Each module is described with its front-end and back-end contributions.

3.1 3.1 bluetooth_handler.c

- **Description**: Manages Bluetooth communication, including device discovery, connection setup, and data transmission.
- **Front-End Role**:
  - Allows users to enable/disable Bluetooth broadcasting or listening.
  - Displays nearby devices for manual connection (optional CLI interface).
- **Back-End Role**:
  - Initializes Bluetooth adapter using BlueZ (e.g., `hci_open_dev`).
  - Scans for nearby devices and listens for incoming packets.
  - Broadcasts messages or file chunks to all discoverable devices.
- **Key Functions**:
  - `init_bluetooth()`: Sets up Bluetooth adapter.
  - `scan_devices()`: Discovers nearby Bluetooth devices.
  - `broadcast_data()`: Sends data packets to all reachable devices.

3.2 3.2 message_manager.c

- **Description**: Handles the creation, transmission, reception, and relaying of alert messages.
- **Front-End Role**:
  - Accepts user input for alert messages (e.g., via CLI: “send alert ’Flood Warning!’”).
  - Displays received alerts to the user.
- **Back-End Role**:
  - Assigns unique message IDs and TTL (time-to-live) to prevent loops.
  - Relays received messages to other devices if TTL > 0.
  - Interfaces with `cache.c` to check for duplicates.
- **Key Functions**:
  - `create_message()`: Generates a message with ID and TTL.
  - `relay_message()`: Forwards messages to other devices.
  - `display_message()`: Shows received messages to the user.

3.3 3.3 file_transfer.c

- **Description**: Manages file splitting, transmission, and reassembly for peer-to-peer file sharing.
- **Front-End Role**:

  - Accepts user input for file selection (e.g., via CLI: “share file.txt”).
  - Notifies users when a file is fully received and reassembled.
- **Back-End Role**:
  - Splits files into small chunks (e.g., 1 KB) for Bluetooth transmission.
  - Assigns chunk IDs and tracks received chunks.
  - Reassembles chunks into the original file at the receiver.
- **Key Functions**:
  - `split_file()`: Divides a file into chunks.
  - `send_chunk()`: Broadcasts a file chunk via Bluetooth.
  - `reassemble_file()`: Combines received chunks into a file.

3.4 3.4 cache.c

- **Description**: Maintains a local cache to store message and file chunk IDs to prevent redundant broadcasts.
- **Front-End Role**:
  - Minimal interaction; may display cache statistics (e.g., number of stored messages).
- **Back-End Role**:
  - Stores message and chunk IDs in a lightweight data structure (e.g., hash table).
  - Checks incoming messages/chunks against the cache to avoid duplicates.
  - Purges old entries based on TTL or cache size limits.
- **Key Functions**:
  - `add_to_cache()`: Stores a new message/chunk ID.
  - `check_duplicate()`: Verifies if a message/chunk was seen before.
  - `clean_cache()`: Removes expired entries.

3.5 3.5 simulator_controller.c

- **Description**: Simulates Bluetooth communication for testing on systems without Bluetooth hardware.
- **Front-End Role**:
  - Allows users to start/stop the simulation via CLI.
  - Displays simulated device interactions (e.g., “Device A sent message to Device B”).
- **Back-End Role**:

  - Emulates Bluetooth via Unix sockets or local message passing.
  - Coordinates timing and message flow between simulated devices.
  - Logs simulation events for debugging.
- **Key Functions**:
  - `start_simulation()`: Initializes the simulation environment.
  - `simulate_broadcast()`: Mimics Bluetooth broadcasting.
  - `log_event()`: Records simulation events for analysis.

4 4. How Modules Work Together

- **Initialization**: `bluetooth_handler.c` (or `simulator_controller.c` in simulation mode) initializes the Bluetooth adapter or socket-based environment.
- **User Interaction**: The user sends an alert or file via CLI, handled by `message_manager.c` or `file_transfer.c`, which prepares the data for transmission.
- **Transmission**: `bluetooth_handler.c` broadcasts the message or file chunks to nearby devices.
- **Relaying**: `message_manager.c` and `file_transfer.c` check with `cache.c` to avoid duplicates, then relay non-duplicate messages/chunks to other devices.
- **Reception**: Receiving devices use `message_manager.c` to display alerts and `file_transfer.c` to reassemble files, with `cache.c` ensuring no redundant processing.
- **Simulation**: If testing, `simulator_controller.c` manages the flow of messages/chunks between simulated devices, mimicking real Bluetooth behavior.

5 5. Use Case Example: Emergency Alert

- User A sends an alert (“Flood Warning!”) via CLI.
- `message_manager.c` creates a message with a unique ID and TTL.
- `bluetooth_handler.c` broadcasts it to nearby devices (B and C).
- Device B checks `cache.c`, confirms the message is new, displays it, and relays it to Device D.
- Device C, having already received the message from B, discards it via `cache.c`.
- The alert propagates until TTL expires, ensuring wide coverage without internet.

6 6. Conclusion

BlueAlert OS is a modular, decentralized system that leverages Bluetooth for alert broadcasting and file sharing. Its five modules (`bluetooth_handler.c`, `message_manager.c`, `file_transfer.c`, `cache.c`, and `simulator_controller.c`) work seamlessly to provide front-end user interaction and robust back-end functionality. The system is ideal for emergency communication or offline scenarios, with a beginner-friendly design that avoids complex kernel modifications.