**EXPLANATION**

### ****Brief Explanation of the Program:****

Our program is an **environmental monitoring system** designed to retrieve real-time weather data (such as temperature and humidity) from an API, process it, log it, and provide alerts based on certain conditions. The program is built using **C programming**, utilizing APIs, JSON parsing, and automated processes.

### ****Key Components and Workflow:****

1. **Fetching Data**:
   * **CURL** is used to fetch weather data from the **OpenWeatherMap API**.
   * The URL (http://api.openweathermap.org/data/2.5/weather?q=Karachi,PK&units=metric&appid=your\_api\_key) is called to retrieve weather data for Karachi, Pakistan.
   * The data is received in **JSON format**.
2. **JSON Parsing**:
   * The program parses the JSON response using the **json-c library**.
   * Extracted information includes:
     + City name
     + Country code
     + Temperature
     + Humidity
3. **Logging Data**:
   * The retrieved and parsed weather data is saved to a file (weather\_data.txt) using the **write\_to\_file** function.
   * The program logs each new set of weather data in the file, appending it to the existing content.
4. **Alerts**:
   * The program checks if certain environmental conditions are met, such as:
     + High temperature (> 30°C)
     + Low humidity (< 20%)
   * If these conditions are met, alerts are triggered using:
     + **syslog** to log alerts at different levels (e.g., LOG\_ALERT for high temperature, LOG\_WARNING for low humidity).
     + **Desktop notifications** are sent via notify-send to alert the user in real-time (for Linux systems).
     + Alerts are printed in the terminal and logged.
5. **Automation**:
   * The program is automated using a **Bash script** with a while loop. It continuously fetches weather data, prints the output in the terminal, saves the data to a file, and alerts the user every 60 seconds.
   * The program's output is logged and displayed using the tee command to capture and display the output simultaneously in the terminal and in the weather\_data.txt file.
6. **Cron Job for Scheduled Execution**:
   * The program can also be automated with **cron** to run at specific times or intervals (e.g., once per day).
   * A cron job is used to run the script automatically at defined times, ensuring the program is executed without manual intervention.

### ****How the System Works:****

* The program fetches weather data from the OpenWeatherMap API, processes the data, and logs it.
* If critical conditions (high temperature or low humidity) are met, alerts are generated in real-time.
* The Bash script handles automation, running the program periodically with a beep sound to indicate activity.
* **Cron jobs** are used to automate the process at regular intervals (e.g., daily or hourly) without manual intervention.

### ****Technologies Used:****

* **C Programming**: For writing the core application.
* **libcurl**: For making HTTP requests to the weather API.
* **json-c**: For parsing the JSON data.
* **syslog**: For logging critical alerts.
* **notify-send**: For desktop notifications on Linux.
* **Bash Scripting**: For automation using a while loop and cron jobs.

This program integrates multiple components to automate weather data collection, processing, logging, and alerting, making it a useful tool for real-time environmental monitoring.

**Explanation of the Weather Fetch Program**

This C program retrieves and processes real-time weather data from an API and outputs the results to the terminal, saves them to a file, and generates alerts for critical conditions. Below is a brief description of its purpose and the functions included:

**Purpose:**

* The program interacts with OpenWeatherMap API to fetch weather data for a specific location.
* It processes the data to extract details like city, country, temperature, and humidity.
* Alerts are generated for high temperature or low humidity using desktop notifications and logged into the system.

**Functions:**

1. **WriteCallback**:
   * A helper function used by the CURL library to write data from the API response into a string buffer.
2. **get\_weather\_data**:
   * Fetches weather data from the API using CURL.
   * Stores the raw JSON response into a string buffer.
   * Prints the raw JSON data to the terminal.
3. **parse\_json**:
   * Parses the JSON response using the json-c library.
   * Extracts weather details such as:
     + City name
     + Country code
     + Temperature
     + Humidity
   * Outputs the parsed details to the terminal.
4. **write\_to\_file**:
   * Saves the weather details (city, country, temperature, and humidity) to a file named weather\_data.txt.
5. **check\_alerts**:
   * Checks for critical conditions:
     + High temperature (>30°C).
     + Low humidity (<20%).
   * Logs alerts to the syslog.
   * Sends desktop notifications for critical conditions.
6. **main**:
   * Allocates memory for storing the API response and weather details.
   * Defines the API URL and calls get\_weather\_data to fetch the weather data.
   * Calls parse\_json to extract relevant details.
   * Uses write\_to\_file to save the data to a file.
   * Invokes check\_alerts to handle notifications for critical conditions.
   * Frees allocated memory before exiting.

This file serves as the primary driver for the environmental monitoring system, integrating data fetching, processing, storage, and alerting functionalities.

**Explanation of the Header File: weather(header).h**

**Purpose:**

This header file provides the structure definition and function declarations used in the weather monitoring system. It ensures modularity by separating the implementation (.c file) from the interface (.h file). Other .c files can include this header to use the functions and the data structure defined here.

**Components:**

1. **Include Guards (#ifndef, #define, #endif)**:
   * Ensures the file is not included multiple times in a single compilation unit, preventing redefinition errors.

c

Copy code

#ifndef WEATHER\_H

#define WEATHER\_H

...

#endif

1. **Structure Definition (WeatherData)**:
   * Defines a WeatherData struct to store the parsed weather details:
     + **city**: Name of the city (string, up to 100 characters).
     + **country**: Country code (string, up to 100 characters).
     + **temperature**: Temperature in degrees Celsius (double).
     + **humidity**: Humidity percentage (integer).

c

Copy code

typedef struct {

char city[100];

char country[100];

double temperature;

int humidity;

} WeatherData;

1. **Function Declarations**:
   * These declarations specify the interface for the functions implemented in the .c file. Each function operates on or interacts with the WeatherData structure.
   * **get\_weather\_data**:
     + Fetches the weather data from the API.
     + Parameters:
       - const char \*url: API endpoint URL.
       - char \*response: Buffer to store the API response.
   * **parse\_json**:
     + Parses the JSON response to extract weather details.
     + Parameters:
       - const char \*json\_data: The raw JSON string.
       - WeatherData \*data: Pointer to the WeatherData structure to populate.
   * **write\_to\_file**:
     + Saves weather details to a file.
     + Parameters:
       - const char \*filename: Name of the file to save data.
       - const WeatherData \*data: Pointer to the weather data.
   * **check\_alerts**:
     + Checks for critical weather conditions and generates alerts.
     + Parameters:
       - const WeatherData \*data: Pointer to the weather data.

**Role in the Program:**

* Defines the WeatherData structure for consistent data representation.
* Ensures all functions have a clear interface, improving readability and maintainability.
* Facilitates reusability and modularity by allowing other .c files to use these functions without redefining them.

### Explanation of the Bash Script: Automating Weather Monitoring with Beep Alerts

#### ****Purpose:****

This script repeatedly runs the weather\_program executable, displaying its output in the terminal, saving it to a file (weather\_data.txt), and producing a beep sound after each iteration. The process runs in an infinite loop and can be terminated using Ctrl + C.

### ****Components of the Script:****

1. **Trap Signal for Graceful Termination:**

bash

Copy code

trap 'echo -e "\nProcess terminated."; break' SIGINT

* + Captures the SIGINT signal (triggered by Ctrl + C).
  + Displays a message (Process terminated.) and breaks the infinite loop, stopping the script gracefully.

1. **User Information:**

bash

Copy code

echo "Press Ctrl + C to stop the repeating beep process."

* + Informs the user how to stop the process.

1. **Infinite Loop:**

bash

Copy code

while true; do

...

done

* + Keeps the script running indefinitely until interrupted by Ctrl + C.

1. **Run the Weather Program and Save Output:**

bash

Copy code

./weather\_program | tee -a weather\_data.txt

* + Executes the weather\_program binary, which generates weather information.
  + Pipes (|) the program's output to the tee command:
    - tee: Displays the output in the terminal while appending it (-a) to the weather\_data.txt file.

1. **Generate a Beep Sound:**

bash

Copy code

echo -e "\a"

* + Sends the bell character (\a) to the terminal, producing a beep sound to alert the user.

1. **Sleep Between Iterations:**

bash

Copy code

sleep 5

* + Waits for 5 seconds before running the program again.
  + This interval can be adjusted based on user preference.

### ****Key Features:****

* **Real-Time Display and Logging**:
  + Ensures that each iteration's output is visible on the terminal and saved to a file for later reference.
* **Beep Alerts**:
  + Provides an audible notification for every new weather data fetch, keeping the user aware.
* **Graceful Exit**:
  + Allows the user to terminate the script cleanly using Ctrl + C.

### ****Use Case in Weather Monitoring System:****

* This script complements the weather monitoring system by automating the repeated execution of the weather program and logging data.
* Ideal for scenarios requiring periodic updates (e.g., every 5 seconds).
* Enhances usability with both terminal alerts and persistent logs.

**Weather\_data.txt**: Stores the data fetched from the program.

**Weather program**: Just runs the program

**Automation with while loop vs. cron:**

1. **Using while loop (Bash Script)**:
   * **Continuous Execution**: The script runs indefinitely until manually terminated (e.g., via Ctrl + C).
   * **Flexible Control**: You can easily modify the interval between iterations (e.g., sleep 5), and you can interact with the script while it's running.
   * **Real-Time Output**: The output appears immediately in the terminal as it executes.
2. **Using cron**:
   * **Scheduled Execution**: cron runs the task at predefined intervals (e.g., every minute, hour, day). It doesn't require user intervention once set up.
   * **Background Process**: cron jobs are managed by the system and run in the background, so you don't see the output unless you specify it to be logged.
   * **No Real-Time Interaction**: cron isn't designed for interactive processes, so it doesn't allow real-time control or interaction while running.

**Key Difference:**

* **Control**: while loop offers continuous and interactive control, while cron is best for running tasks at fixed times or intervals without constant supervision.
* **Use Case**: Use a while loop for real-time monitoring or when you need the script to run continuously. Use cron when you need periodic tasks or background execution that doesn't require interaction.