

Lab 6

Part 1 Process

This was relatively straight forward. I had ChatGPT build the web server in python per the lab instructions. I then created a location.txt file in the same directory so that clients have an object that they can request. After that it was just a matter of bug fixing it using the curl commands and ChatGPT to get it to work. The ChatGPT information is shown in the sources below.

Part 2 Process

I started off with the http_request example for this. I then used ChatGPT to convert the get request into a post request function. After that I simply copied my lab 2 code for the temperature and humidity sensors over. In the post task function while loop I used the Lab 2 code to get the current humidity and temperature. I then inserted them into the post request and let the post function handle the rest.

Sources



For Lab 6.1 you will be setting up a customized Web Server running with in the IP address space of your Local Area Network. Python is a good language to use to do this. We recommend getting help from ChatGPT to get there quickly. We will refer to the this web server as the “Weather Monitor”. It will perform two tasks, which will both use the port: 1234. You will also set up a HotSpot that will allow the ESP32C3 peripheral (the Weather Station) to have WiFi access to the Internet (WAN), and to the WebServer (LAN). You will have to make that the HotSpot can access the IP addresses in your local area network. We will refer to the ESP32C3 as the “Weather Station”. The “Weather Station” will be directly accessing the website: www.wttr.in in order to get access to outdoor weather information at numerous locations in the World. The Hotspot allows the “Weather Station” (ESP32C3) to communicate with both the Internet, and the Weather Monitor (web server) accessible from your Local Area Network. We will test that this all works via the use of the program curl, performing HTTP GET and POST requests based on the following commands: `weather_server=192.168.137.129:1234 curl http://$weather_server/readfile/location.txt` This should return a string with the location found in file: location.txt That file should contain something like: “Scotts+Valley”. Curl should return the contents of that file. You will have to customize your web server so this works. You will also use an HTTP POST to have your mini-web server output a string. By running the following script. `1 weather_server=192.168.137.129:1234 location="Scotts Valley" contents="$location: Outdoors T:74F, H:36%, Indoors T:70F, H:40%\n" curl -X POST -H "Content-Type: text/plain; charset=utf-8" \ -d "$contents" $weather_server/weather` The above HTTP POST will cause the “Weather Monitor” to

generate output that looks something like this: "Scotts Valley: Outdoors T:47F, H:36%, Indoors T:70F, H:40%" The \n should provide a line feed, so that it can be done multiple times, one line after the other. Finally running the next command should access the remote web site www.wttr.in and get the outdoor weather parameters from whatever location you specify. Here we are specifying the city of Scotts Valley. loc="Scotts+Valley" curl https://www.wttr.in/\$loc?format=%l:+%t+%h echo It should generate output like this: Scotts+Valley: +50°F 54% Note: Try running the verbose version of these curl commands by using the -v flag. This will show you the headers that are being sent to the various http servers.

ChatGPT

It seems like you have a comprehensive set of tasks to accomplish for Lab 6.1, involving setting up a customized web server ("Weather Monitor") using Python, configuring a HotSpot for ESP32C3 ("Weather Station"), and testing the system using curl commands for HTTP GET and POST requests. Let's break down the tasks step by step.

Task 1: Set up the Weather Monitor (Customized Web Server)

1. Create a Python script for the web server:

python

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/readfile/location.txt', methods=['GET'])
def read_location_file():
    # Customize this part to read the location from a file
    with open('location.txt', 'r') as file:
        location = file.read()
    return location

@app.route('/weather', methods=['POST'])
def update_weather():
    # Customize this part to handle the POST request and
    generate the response
    data = request.data.decode('utf-8')
    # Process data and generate response
```

```
    return data

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=1234)
```

1.

2. **Run the script:**

Save the script with a `.py` extension (e.g., `weather_monitor.py`) and run it.
This will start your web server on the specified IP address and port.

Task 2: Set up the HotSpot for ESP32C3

Configure a HotSpot to allow ESP32C3 to connect to both the Internet and the local web server. Ensure that the HotSpot has access to your local area network.

Task 3: Test with curl commands

Now, you can test the setup using the provided curl commands.

1. **HTTP GET request to read location file:**

bash

```
weather_server=192.168.137.129:1234
curl http://$weather_server/readfile/location.txt
```

Make sure this returns the location from the `location.txt` file.

HTTP POST request to update weather:

bash

```
weather_server=192.168.137.129:1234
location="Scotts Valley"
contents="$location: Outdoors T:74F, H:36%, Indoors T:70F,
H:40%\n"
curl -X POST -H "Content-Type: text/plain; charset=utf-8" -d
"$contents" $weather_server/weather
```

Ensure this updates the weather information on the server.

Access the remote website for outdoor weather parameters:

bash

```
loc="Scotts+Valley"
curl https://www.wttr.in/$loc?format=%l:+%t+%h
echo
```

1. Verify that this command retrieves and displays the outdoor weather parameters for Scotts Valley.
2. **Optional: Run verbose versions of curl commands:**
Add the `-v` flag to your curl commands for more detailed information.

Remember to adapt the Python script based on your specific requirements for reading the location file and handling the POST request.



Can you modify the code to handle the POST request

ChatGPT

Certainly! I'll modify the code to handle the POST request and generate the appropriate response for the "Weather Monitor" server.

```
python
from flask import Flask, request

app = Flask(__name__)

# Initialize location variable
location = ""

@app.route('/readfile/location.txt', methods=['GET'])
def read_location_file():
    return location

@app.route('/weather', methods=['POST'])
def update_weather():
    global location
    data = request.data.decode('utf-8')

    # Assuming the data format is "<location>: <outdoor info>\n"
    parts = data.split(':')
    if len(parts) == 2:
        location = parts[0].strip()
        return f"Location updated to: {location}"
    else:
```

```
        return "Invalid data format"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=1234)
```

In this modified code:

- The `location` variable is declared as global to store the location data.
- The `/readfile/location.txt` endpoint returns the current location stored in the `location` variable.
- The `/weather` endpoint handles the POST request, updates the `location` variable, and returns a response.

Make sure to customize the data processing in the `update_weather` function based on the actual format of the data you expect in the POST request. This example assumes a simple format where the data is "<location>: <outdoor info>\n". Adjust it according to your needs.



```
# Python 3 server example from http.server import BaseHTTPRequestHandler,
HTTPServer import time hostName = "Weather Monitor" serverPort = 1234 class
MyServer(BaseHTTPRequestHandler): def do_GET(self): self.send_response(200)
self.send_header("Content-type", "text/html") self.end_headers() f = open("location.txt",
"r") s = f.read() self.wfile.write(s) if __name__ == "__main__": webServer =
HTTPServer((hostName, serverPort), MyServer) print("Server started http://%s:%s" %
(hostName, serverPort)) try: webServer.serve_forever() except KeyboardInterrupt: pass
webServer.server_close() print("Server stopped.")
```

ChatGPT

It looks like you've provided a simple Python 3 server example using the `http.server` module to handle GET requests and serve the content of the "location.txt" file. If you want to extend this server to handle a POST request as well, you can modify the `do_GET` method to check for the request method and handle both GET and POST accordingly.

Here's an updated version of your code to handle both GET and POST requests:

```
python
from http.server import BaseHTTPRequestHandler, HTTPServer
```

```
import time

hostName = "Weather Monitor"
serverPort = 1234

class MyServer(BaseHTTPRequestHandler):
    def do_GET(self):
        if self.path == "/readfile/location.txt":
            self.send_response(200)
            self.send_header("Content-type", "text/html")
            self.end_headers()
            with open("location.txt", "r") as file:
                content = file.read()
                self.wfile.write(content.encode("utf-8"))
        else:
            self.send_response(404)
            self.end_headers()
            self.wfile.write(b"Not Found")

    def do_POST(self):
        if self.path == "/weather":
            content_length = int(self.headers['Content-Length'])
            post_data =
self.rfile.read(content_length).decode("utf-8")
            # Assuming the data format is "<location>: <outdoor
info>\n"

            parts = post_data.split(':')
            if len(parts) == 2:
                location = parts[0].strip()
                # Process and store the location data as needed
                self.send_response(200)
                self.send_header("Content-type", "text/html")
                self.end_headers()
                self.wfile.write(f"Location updated to:
{location}".encode("utf-8"))
```

```

        else:
            self.send_response(400)
            self.end_headers()
            self.wfile.write(b"Invalid data format")
    else:
        self.send_response(404)
        self.end_headers()
        self.wfile.write(b"Not Found")

if __name__ == "__main__":
    webServer = HTTPServer((hostName, serverPort), MyServer)
    print("Server started http://%s:%s" % (hostName,
serverPort))

    try:
        webServer.serve_forever()
    except KeyboardInterrupt:
        pass

webServer.server_close()
print("Server stopped.")

```

This modification introduces a `do_POST` method to handle POST requests. It checks if the request path is `"/weather"` and processes the data accordingly. Adjust the data processing logic based on your actual data format and requirements.



```
File "c:\Users\bradl\Dropbox\Documents\cse_121\webserver.py", line 44, in <module>
webServer = HTTPServer((hostName, serverPort), MyServer)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File
"C:\Users\bradl\anaconda3\Lib\socketserver.py", line 456, in __init__ self.server_bind()
File "C:\Users\bradl\anaconda3\Lib\http\server.py", line 136, in server_bind
socketserver.TCPServer.server_bind(self) File
"C:\Users\bradl\anaconda3\Lib\socketserver.py", line 472, in server_bind
self.socket.bind(self.server_address) socket.gaierror: [Errno 11001] getaddrinfo failed
```

It looks like the issue is related to the `hostName` variable when creating the `HTTPServer` instance. The `HTTPServer` expects a tuple with the server address, and it seems like you might have set `hostName` to a string that doesn't represent a valid IP address or hostname.

Instead of using a string for `hostName`, you can set it to an empty string (`""`) to bind to all available interfaces, like this:

```
python
hostName = ""
```

So, your code should look like this:

```
python
from http.server import BaseHTTPRequestHandler, HTTPServer
import time
```

```
hostName = "" # Change this line
serverPort = 1234
```

```
class MyServer(BaseHTTPRequestHandler):
    def do_GET(self):
        if self.path == "/readfile/location.txt":
            self.send_response(200)
            self.send_header("Content-type", "text/html")
            self.end_headers()
            with open("location.txt", "r") as file:
                content = file.read()
                self.wfile.write(content.encode("utf-8"))
        else:
            self.send_response(404)
            self.end_headers()
            self.wfile.write(b"Not Found")

    def do_POST(self):
        if self.path == "/weather":
```



```
        content_length = int(self.headers['Content-Length'])
        post_data =
self.rfile.read(content_length).decode("utf-8")
        # Assuming the data format is "<location>: <outdoor
info>\n"

        parts = post_data.split(':')
        if len(parts) == 2:
            location = parts[0].strip()
            # Process and store the location data as needed
            self.send_response(200)
            self.send_header("Content-type", "text/html")
            self.end_headers()
            self.wfile.write(f"Location updated to:
{location}".encode("utf-8"))
        else:
            self.send_response(400)
            self.end_headers()
            self.wfile.write(b"Invalid data format")
    else:
        self.send_response(404)
        self.end_headers()
        self.wfile.write(b"Not Found")

if __name__ == "__main__":
    webServer = HTTPServer((hostName, serverPort), MyServer)
    print("Server started http://%s:%s" % (hostName,
serverPort))

    try:
        webServer.serve_forever()
    except KeyboardInterrupt:
        pass

    webServer.server_close()
    print("Server stopped.")
```

This change should resolve the `socket.gaierror: [Errno 11001] getaddrinfo failed` issue.



```
from flask import Flask, request
app = Flask(__name__) # Initialize location variable
location = ""
@app.route('/readfile/location.txt', methods=['GET'])
def read_location_file():
    f = open("location.txt", "r")
    location = f.read()
    return location
@app.route('/weather', methods=['POST'])
def update_weather():
    global location
    data = request.data.decode('utf-8')
    # Assuming the data format is "<location>: <outdoor info>\n"
    parts = data.split(':')
    if len(parts) == 2:
        location = parts[0].strip()
    return f"Location updated to: {location}"
else:
    return "Invalid data format"
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=1234)
```

ChatGPT

Your Flask code looks good for setting up a simple web server with two routes: one for handling GET requests to read the contents of "location.txt" and another for handling POST requests to update the location.

Here's a slight improvement to avoid potential issues with file handling (closing the file properly):

```
python
from flask import Flask, request

app = Flask(__name__)

# Initialize location variable
location = ""

@app.route('/readfile/location.txt', methods=['GET'])
def read_location_file():
    with open("location.txt", "r") as file:
        location = file.read()
    return location

@app.route('/weather', methods=['POST'])
def update_weather():
```

```
global location
data = request.data.decode('utf-8')

# Assuming the data format is "<location>: <outdoor info>\n"
parts = data.split(':')
if len(parts) == 2:
    location = parts[0].strip()
    return f"Location updated to: {location}"
else:
    return "Invalid data format"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=1234)
```

In this modification, the `with open("location.txt", "r") as file` context manager is used to ensure that the file is properly closed after reading its contents. This is a good practice to prevent potential issues with file handling.

Make sure to test your web server with both GET and POST requests to verify that it behaves as expected.

[illegible]

ChatGPT

The error `FileNotFoundError: [Errno 2] No such file or directory: 'location.txt'` indicates that the Flask application is unable to find the "location.txt" file. Make sure that the file exists in the same directory as your Flask script or provide an absolute path to the file.

Here are a few steps to troubleshoot and resolve the issue:

1. **Check the File Path:**

- Ensure that "location.txt" is in the same directory as your Flask script.
- Double-check the spelling and case sensitivity of the file name.

2. **Provide an Absolute Path:**

- Instead of relying on the current working directory, provide an absolute path to the file:

python

```
import os
```

```
# Get the absolute path of the script's directory
script_dir = os.path.dirname(os.path.realpath(__file__))
file_path = os.path.join(script_dir, "location.txt")
```

```
with open(file_path, "r") as file:
    location = file.read()
```

○

2. **Check File Permissions:**

- Ensure that the script has read permissions for the "location.txt" file.
- Make sure the file is not accidentally deleted or moved.

Make these adjustments, and the error should be resolved. If you continue to encounter issues, please provide more details about the file's location and your directory structure.



I tried these commands in a linux terminal "location="Scotts Valley" contents="\$location: Outdoors T:74F, H:36%, Indoors T:70F, H:40%\n" curl -X POST -H "Content-Type: text/plain; charset=utf-8" \ -d "\$contents" \$weather_server/weather" and it gave me the error "Invalid data format"

ChatGPT

The "Invalid data format" error indicates that the server received a POST request to the `/weather` endpoint, but the data format in the request body did not match the expected format in the server code.

In your Python server code, you are assuming that the data format in the POST request is "`<location>: <outdoor info>\n`". Therefore, when you run the curl command, ensure that the data follows this format.

Here is your curl command:

```
bash
location="Scotts Valley"
contents="$location: Outdoors T:74F, H:36%, Indoors T:70F, H:40%\n"
curl -X POST -H "Content-Type: text/plain; charset=utf-8" \
-d "$contents" $weather_server/weather
```

Make sure that the content of the `$contents` variable matches the expected format. In this case, it should be "Scotts Valley: Outdoors T:74F, H:36%, Indoors T:70F, H:40%\n".

If the problem persists, you may want to add some debugging statements to your server code to print or log the received data in the `update_weather` function. This can help you identify the exact format of the data being received by the server. For example:

```
python
@app.route('/weather', methods=['POST'])
def update_weather():
    global location
    data = request.data.decode('utf-8')

    # Debugging statement
    print(f"Received data: {data}")

    # Assuming the data format is "<location>: <outdoor info>\n"
    parts = data.split(':')
    if len(parts) == 2:
        location = parts[0].strip()
        return f"Location updated to: {location}"
```

```
else:  
    return "Invalid data format"
```

By printing the received data, you can better understand why the server is considering it as an invalid format. Adjust the client-side curl command accordingly.



```
This is the webserver code: from flask import Flask, request import os app =  
Flask(__name__) # Initialize location variable location = ""  
@app.route('/readfile/location.txt', methods=['GET']) def read_location_file(): script_dir =  
os.path.dirname(os.path.realpath(__file__)) file_path = os.path.join(script_dir,  
"location.txt") with open(file_path, "r") as file: location = file.read() return location  
@app.route('/weather', methods=['POST']) def update_weather(): global location data =  
request.data.decode('utf-8') # Assuming the data format is "<location>: <outdoor  
info>\n" parts = data.split(':') if len(parts) == 2: location = parts[0].strip() return f"Location  
updated to: {location}" else: return "Invalid data format" if __name__ == '__main__':  
app.run(host='0.0.0.0', port=1234)
```

ChatGPT

Your Flask code looks correct, and it should work as expected for the provided curl command:

```
bash  
location="Scotts Valley"  
contents="$location: Outdoors T:74F, H:36%, Indoors T:70F,  
H:40%\n"  
curl -X POST -H "Content-Type: text/plain; charset=utf-8" \  
-d "$contents" $weather_server/weather
```

However, I noticed that you are using the variable `$weather_server` in your curl command, and it seems like you might not have defined or assigned a value to it. Ensure that `$weather_server` holds the correct IP address and port where your Flask server is running.

If you are running the Flask server locally, you can set `$weather_server` to `localhost` or `127.0.0.1`. If the Flask server is on another machine, set it to the appropriate IP address or hostname of that machine.

For example:

```
bash
weather_server="127.0.0.1:1234"
location="Scotts Valley"
contents="$location: Outdoors T:74F, H:36%, Indoors T:70F,
H:40%\n"
curl -X POST -H "Content-Type: text/plain; charset=utf-8" \
-d "$contents" http://$weather_server/weather
```

Make sure to replace "127.0.0.1:1234" with the correct value for your server.

If you are still facing issues, you may want to print or log the received data in the `update_weather` function, as mentioned earlier, to debug the format of the data being sent to the server.



This is the instructions for the second part of the lab: "Create your "UCSC Weather Station" HTTP client application. The ESP32C3 reads the humidity and temperature sensor like you have done in the past and it will perform an HTTP Post against the IP address of the "Weather Monitor" (i.e. your customized web server) while using port 1234. This will be using an IP address in a local area network accessible by your hotspot. The ESP32C3 should do an HTTP POST with the Temperature and Humidity to port 1234. Similar to the Post as shown in the curl command given in part 6.1. Once the post is realized. The "UCSC Weather Service" (web server) should display the information by just printing it out (you do not need to display it through a browser). Hint: look for the example: `http_request` under the protocols and examples subdirectory for how to do this with the Espressif libraries." This is my code so far: /* HTTP GET Example using plain POSIX sockets This example code is in the Public Domain (or CC0 licensed, at your option.) Unless required by applicable law or agreed to in writing, this software is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. */ #include <string.h> #include "freertos/FreeRTOS.h" #include "freertos/task.h" #include "esp_system.h" #include "esp_wifi.h" #include "esp_event.h" #include "esp_log.h" #include "nvs_flash.h" #include "protocol_examples_common.h" #include "lwip/err.h" #include "lwip/sockets.h" #include "lwip/sys.h" #include "lwip/netdb.h" #include "lwip/dns.h" #include "sdkconfig.h" #include <stdio.h> #include <inttypes.h> #include "driver/i2c.h" #define I2C_MASTER_SCL_IO 8 #define I2C_MASTER_SDA_IO 10 #define I2C_MASTER_NUM I2C_NUM_0 #define I2C_MASTER_FREQ_HZ 400000 /* Constants that aren't configurable in menuconfig */

```

#define WEB_SERVER "example.com" #define WEB_PORT "80" #define WEB_PATH
"/" static const char *TAG = "example"; static const char *REQUEST = "GET "
WEB_PATH " HTTP/1.0\r\n" "Host: "WEB_SERVER":"WEB_PORT"\r\n" "User-Agent:
esp-idf/1.0 esp32\r\n" "\r\n"; uint8_t shtc3_read_temperature(uint16_t* temperature) {
uint8_t data[2] = {0x78, 0x66}; i2c_cmd_handle_t cmd = i2c_cmd_link_create();
i2c_master_start(cmd); i2c_master_write_byte(cmd, (SHTC3_ADDR << 1) |
I2C_MASTER_WRITE, true); i2c_master_write(cmd, data, 2, true);
i2c_master_stop(cmd); esp_err_t ret = i2c_master_cmd_begin(I2C_MASTER_NUM,
cmd, 1000 / portTICK_PERIOD_MS); i2c_cmd_link_delete(cmd); if (ret != ESP_OK) {
return ret; } vTaskDelay(20 / portTICK_PERIOD_MS); // Wait for the measurement to
complete uint8_t temperature_data[6]; cmd = i2c_cmd_link_create();
i2c_master_start(cmd); i2c_master_write_byte(cmd, (SHTC3_ADDR << 1) |
I2C_MASTER_READ, true); i2c_master_read(cmd, temperature_data, 6,
I2C_MASTER_LAST_NACK); i2c_master_stop(cmd); ret =
i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, 1000 / portTICK_PERIOD_MS);
i2c_cmd_link_delete(cmd); if (ret != ESP_OK) { return ret; } *temperature =
(temperature_data[0] << 8) | temperature_data[1]; return ESP_OK; } uint8_t
shtc3_read_humidity(uint16_t* humidity) { uint8_t data[2] = {0x78, 0x66};
i2c_cmd_handle_t cmd = i2c_cmd_link_create(); i2c_master_start(cmd);
i2c_master_write_byte(cmd, (SHTC3_ADDR << 1) | I2C_MASTER_WRITE, true);
i2c_master_write(cmd, data, 2, true); i2c_master_stop(cmd); esp_err_t ret =
i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, 1000 / portTICK_PERIOD_MS);
i2c_cmd_link_delete(cmd); if (ret != ESP_OK) { return ret; } vTaskDelay(20 /
portTICK_PERIOD_MS); // Wait for the measurement to complete uint8_t
humidity_data[6]; cmd = i2c_cmd_link_create(); i2c_master_start(cmd);
i2c_master_write_byte(cmd, (SHTC3_ADDR << 1) | I2C_MASTER_READ, true);
i2c_master_read(cmd, humidity_data, 6, I2C_MASTER_LAST_NACK);
i2c_master_stop(cmd); ret = i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, 1000 /
portTICK_PERIOD_MS); i2c_cmd_link_delete(cmd); if (ret != ESP_OK) { return ret; }
*humidity = (humidity_data[3] << 8) | humidity_data[4]; return ESP_OK; } static void
http_get_task(void *pvParameters) { const struct addrinfo hints = { .ai_family =
AF_INET, .ai_socktype = SOCK_STREAM, }; struct addrinfo *res; struct in_addr *addr;
int s, r; char recv_buf[64]; while(1) { int err = getaddrinfo(WEB_SERVER, WEB_PORT,
&hints, &res); if(err != 0 || res == NULL) { ESP_LOGE(TAG, "DNS lookup failed err=%d
res=%p", err, res); vTaskDelay(1000 / portTICK_PERIOD_MS); continue; } /* Code to
print the resolved IP. Note: inet_ntoa is non-reentrant, look at ipaddr_ntoa_r for "real"
code */ addr = &((struct sockaddr_in *)res->ai_addr)->sin_addr; ESP_LOGI(TAG, "DNS
lookup succeeded. IP=%s", inet_ntoa(*addr)); s = socket(res->ai_family,
res->ai_socktype, 0); if(s < 0) { ESP_LOGE(TAG, "... Failed to allocate socket.");
freeaddrinfo(res); vTaskDelay(1000 / portTICK_PERIOD_MS); continue; }

```



```

ESP_LOGI(TAG, "... allocated socket"); if(connect(s, res->ai_addr, res->ai_addrlen) !=
0) { ESP_LOGE(TAG, "... socket connect failed errno=%d", errno); close(s);
freeaddrinfo(res); vTaskDelay(4000 / portTICK_PERIOD_MS); continue; }
ESP_LOGI(TAG, "... connected"); freeaddrinfo(res); if (write(s, REQUEST,
strlen(REQUEST)) < 0) { ESP_LOGE(TAG, "... socket send failed"); close(s);
vTaskDelay(4000 / portTICK_PERIOD_MS); continue; } ESP_LOGI(TAG, "... socket
send success"); struct timeval receiving_timeout; receiving_timeout.tv_sec = 5;
receiving_timeout.tv_usec = 0; if (setsockopt(s, SOL_SOCKET, SO_RCVTIMEO,
&receiving_timeout, sizeof(receiving_timeout)) < 0) { ESP_LOGE(TAG, "... failed to set
socket receiving timeout"); close(s); vTaskDelay(4000 / portTICK_PERIOD_MS);
continue; } ESP_LOGI(TAG, "... set socket receiving timeout success"); /* Read HTTP
response */ do { bzero(recv_buf, sizeof(recv_buf)); r = read(s, recv_buf,
sizeof(recv_buf)-1); for(int i = 0; i < r; i++) { putchar(recv_buf[i]); } } while(r > 0);
ESP_LOGI(TAG, "... done reading from socket. Last read return=%d errno=%d.", r,
errno); close(s); for(int countdown = 10; countdown >= 0; countdown--) {
ESP_LOGI(TAG, "%d... ", countdown); vTaskDelay(1000 / portTICK_PERIOD_MS); }
ESP_LOGI(TAG, "Starting again!"); } } void startI2C() { i2c_config_t conf = { .mode =
I2C_MODE_MASTER, .sda_io_num = I2C_MASTER_SDA_IO, .sda_pullup_en =
GPIO_PULLUP_ENABLE, .scl_io_num = I2C_MASTER_SCL_IO, .scl_pullup_en =
GPIO_PULLUP_ENABLE, .master.clk_speed = I2C_MASTER_FREQ_HZ, };
i2c_param_config(I2C_MASTER_NUM, &conf); i2c_driver_install(I2C_MASTER_NUM,
conf.mode, 0, 0, 0); } void app_main(void) { startI2C(); ESP_ERROR_CHECK(
nvs_flash_init() ); ESP_ERROR_CHECK(esp_netif_init());
ESP_ERROR_CHECK(esp_event_loop_create_default()); /* This helper function
configures Wi-Fi or Ethernet, as selected in menuconfig. * Read "Establishing Wi-Fi or
Ethernet Connection" section in * examples/protocols/README.md for more information
about this function. */ ESP_ERROR_CHECK(example_connect()); while (1) { uint16_t
temperature, humidity; if (shtc3_read_temperature(&temperature) == ESP_OK &&
shtc3_read_humidity(&humidity) == ESP_OK) { float temperature_fahrenheit =
.45+175*(temperature / 65535.0); float temperature_celsius = ((.45+175*(temperature /
65535.0)) - 32) * (0.56); float humidity_percentage = 100* (humidity / 65535.0);
printf("Temperature is %.2fC (or %.2fF) with a %.2f%% humidity\n",
temperature_celsius, temperature_fahrenheit, humidity_percentage); } else {
printf("Failed to read sensor data\n"); } vTaskDelay(pdMS_TO_TICKS(2000)); //
2-second delay } xTaskCreate(&http_get_task, "http_get_task", 4096, NULL, 5, NULL); }

```

ChatGPT

It looks like you've made good progress on the ESP32C3 code to read sensor data and perform an HTTP POST request to your server. Here are a few suggestions and improvements:

1. **Ensure HTTP POST Payload:**

- The HTTP POST request payload sent by your ESP32C3 should match the expected format on the server side. In your Python server code, you are expecting a format like "<location>: <outdoor info>\n". Make sure the payload you are sending from the ESP32C3 adheres to this format.

2. **Update the HTTP Request:**

- Modify the `REQUEST` string to include the proper HTTP POST request with the sensor data. You may need to create a new string dynamically based on the sensor readings.

3. c

```
char request_buffer[128];
snprintf(request_buffer, sizeof(request_buffer), "POST /weather
HTTP/1.0\r\n"
                                           "Host: %s\r\n"
                                           "Content-Type:
application/x-www-form-urlencoded\r\n"
                                           "Content-Length:
%d\r\n\r\n"

"temperature=%.2f&humidity=%.2f",
    WEB_SERVER, (int)(strlen("temperature=&humidity=") + 2
* sizeof(float)),
    temperature_celsius, humidity_percentage);

if (write(s, request_buffer, strlen(request_buffer)) < 0) {
    ESP_LOGE(TAG, "... socket send failed");
    close(s);
    vTaskDelay(4000 / portTICK_PERIOD_MS);
    continue;
}
```

1. This assumes that your server expects the data in the format of a URL-encoded form.

2. **Error Handling:**

- Add more error handling and logging to your code, especially when dealing with I2C communication, socket operations, and HTTP requests. This will help in identifying and diagnosing any issues that may arise.

3. Delay Between HTTP POST Requests:

- Consider adding a delay between successive HTTP POST requests to avoid flooding the server with requests. You have a delay of 2 seconds in the main loop, but ensure that the HTTP POST task has sufficient delay as well.

4. Close the Socket After Reading:

- After reading from the socket, close it to free up resources. Currently, you are only closing the socket if there is an error during reading.

5. Cleanup:

- Organize your code into functions to improve readability and maintainability. For example, move the HTTP POST code into a separate function.

Remember to adapt the code to match the exact requirements and expectations of your server. If you face any specific issues or have more detailed requirements, feel free to ask for further assistance!



Can you change this to be a post instead of a get: static void http_get_task(void *pvParameters) { const struct addrinfo hints = { .ai_family = AF_INET, .ai_socktype = SOCK_STREAM, }; struct addrinfo *res; struct in_addr *addr; int s, r; char recv_buf[64]; while(1) { int err = getaddrinfo(WEB_SERVER, WEB_PORT, &hints, &res); if(err != 0 || res == NULL) { ESP_LOGE(TAG, "DNS lookup failed err=%d res=%p", err, res); vTaskDelay(1000 / portTICK_PERIOD_MS); continue; } /* Code to print the resolved IP. Note: inet_ntoa is non-reentrant, look at ipaddr_ntoa_r for "real" code */ addr = &((struct sockaddr_in *)res->ai_addr)->sin_addr; ESP_LOGI(TAG, "DNS lookup succeeded. IP=%s", inet_ntoa(*addr)); s = socket(res->ai_family, res->ai_socktype, 0); if(s < 0) { ESP_LOGE(TAG, "... Failed to allocate socket."); freeaddrinfo(res); vTaskDelay(1000 / portTICK_PERIOD_MS); continue; } ESP_LOGI(TAG, "... allocated socket"); if(connect(s, res->ai_addr, res->ai_addrlen) != 0) { ESP_LOGE(TAG, "... socket connect failed errno=%d", errno); close(s); freeaddrinfo(res); vTaskDelay(4000 / portTICK_PERIOD_MS); continue; } ESP_LOGI(TAG, "... connected"); freeaddrinfo(res); if (write(s, REQUEST, strlen(REQUEST)) < 0) { ESP_LOGE(TAG, "... socket send failed"); close(s); vTaskDelay(4000 / portTICK_PERIOD_MS); continue; } ESP_LOGI(TAG, "... socket send success"); struct timeval receiving_timeout; receiving_timeout.tv_sec = 5; receiving_timeout.tv_usec = 0; if (setsockopt(s, SOL_SOCKET, SO_RCVTIMEO, &receiving_timeout, sizeof(receiving_timeout)) < 0) { ESP_LOGE(TAG, "... failed to set socket receiving timeout"); close(s); vTaskDelay(4000 / portTICK_PERIOD_MS); continue; } ESP_LOGI(TAG, "... set socket receiving timeout success"); /* Read HTTP response */ do { bzero(recv_buf, sizeof(recv_buf)); r = read(s,

```
recv_buf, sizeof(recv_buf)-1); for(int i = 0; i < r; i++) { putchar(recv_buf[i]); } } while(r > 0); ESP_LOGI(TAG, "... done reading from socket. Last read return=%d errno=%d.", r, errno); close(s); for(int countdown = 10; countdown >= 0; countdown--) { ESP_LOGI(TAG, "%d... ", countdown); vTaskDelay(1000 / portTICK_PERIOD_MS); } ESP_LOGI(TAG, "Starting again!"); } }
```

ChatGPT

Certainly! To modify the code to perform an HTTP POST instead of a GET, you'll need to update the request format. Here's a modified version of your [http_get_task](#) function to perform an HTTP POST:

```
c
static void http_post_task(void *pvParameters)
{
    const struct addrinfo hints = {
        .ai_family = AF_INET,
        .ai_socktype = SOCK_STREAM,
    };
    struct addrinfo *res;
    struct in_addr *addr;
    int s, r;
    char recv_buf[64];

    while (1) {
        int err = getaddrinfo(WEB_SERVER, WEB_PORT, &hints,
&res);

        if (err != 0 || res == NULL) {
            ESP_LOGE(TAG, "DNS lookup failed err=%d res=%p",
err, res);
            vTaskDelay(1000 / portTICK_PERIOD_MS);
            continue;
        }

        /* Code to print the resolved IP.
        Note: inet_ntoa is non-reentrant, look at
ipaddr_ntoa_r for "real" code */
```

```
addr = &((struct sockaddr_in *)res->ai_addr)->sin_addr;
ESP_LOGI(TAG, "DNS lookup succeeded. IP=%s",
inet_ntoa(*addr));

s = socket(res->ai_family, res->ai_socktype, 0);
if (s < 0) {
    ESP_LOGE(TAG, "... Failed to allocate socket.");
    freeaddrinfo(res);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
    continue;
}
ESP_LOGI(TAG, "... allocated socket");

if (connect(s, res->ai_addr, res->ai_addrlen) != 0) {
    ESP_LOGE(TAG, "... socket connect failed errno=%d",
errno);
    close(s);
    freeaddrinfo(res);
    vTaskDelay(4000 / portTICK_PERIOD_MS);
    continue;
}

ESP_LOGI(TAG, "... connected");
freeaddrinfo(res);

// Modified request for HTTP POST
const char *post_request = "POST /weather HTTP/1.0\r\n"
    "Host: " WEB_SERVER ":"
WEB_PORT "\r\n"
    "Content-Type:
application/x-www-form-urlencoded\r\n"
    "Content-Length: %d\r\n\r\n"

"temperature=%.2f&humidity=%.2f";
```

```
    char request_buffer[128];
    snprintf(request_buffer, sizeof(request_buffer),
post_request,
                (int)(strlen("temperature=&humidity=") + 2 *
sizeof(float)),
                temperature_celsius, humidity_percentage);

    if (write(s, request_buffer, strlen(request_buffer)) <
0) {
        ESP_LOGE(TAG, "... socket send failed");
        close(s);
        vTaskDelay(4000 / portTICK_PERIOD_MS);
        continue;
    }
    ESP_LOGI(TAG, "... socket send success");

    struct timeval receiving_timeout;
    receiving_timeout.tv_sec = 5;
    receiving_timeout.tv_usec = 0;
    if (setsockopt(s, SOL_SOCKET, SO_RCVTIMEO,
&receiving_timeout,
                sizeof(receiving_timeout)) < 0) {
        ESP_LOGE(TAG, "... failed to set socket receiving
timeout");
        close(s);
        vTaskDelay(4000 / portTICK_PERIOD_MS);
        continue;
    }
    ESP_LOGI(TAG, "... set socket receiving timeout
success");

    /* Read HTTP response */
    do {
        bzero(recv_buf, sizeof(recv_buf));
        r = read(s, recv_buf, sizeof(recv_buf) - 1);
```

```
        for (int i = 0; i < r; i++) {  
            putchar(recv_buf[i]);  
        }  
    } while (r > 0);  
  
    ESP_LOGI(TAG, "... done reading from socket. Last read  
return=%d errno=%d.", r, errno);  
    close(s);  
    for (int countdown = 10; countdown >= 0; countdown--) {  
        ESP_LOGI(TAG, "%d... ", countdown);  
        vTaskDelay(1000 / portTICK_PERIOD_MS);  
    }  
    ESP_LOGI(TAG, "Starting again!");  
}  
}
```

This code now sends an HTTP POST request with temperature and humidity data to the specified server and port. Adjust the request format as needed based on your server's expectations.



```
/* HTTP GET Example using plain POSIX sockets This example code is in the Public  
Domain (or CC0 licensed, at your option.) Unless required by applicable law or agreed  
to in writing, this software is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES  
OR CONDITIONS OF ANY KIND, either express or implied. */ #include <string.h>  
#include "freertos/FreeRTOS.h" #include "freertos/task.h" #include "esp_system.h"  
#include "esp_wifi.h" #include "esp_event.h" #include "esp_log.h" #include "nvs_flash.h"  
#include "protocol_examples_common.h" #include "lwip/err.h" #include "lwip/sockets.h"  
#include "lwip/sys.h" #include "lwip/netdb.h" #include "lwip/dns.h" #include "sdkconfig.h"  
#include <stdio.h> #include <inttypes.h> #include "driver/i2c.h" #define  
I2C_MASTER_SCL_IO 8 #define I2C_MASTER_SDA_IO 10 #define  
I2C_MASTER_NUM I2C_NUM_0 #define I2C_MASTER_FREQ_HZ 400000 #define  
SHTC3_ADDR 0x70 /* Constants that aren't configurable in menuconfig */ #define  
WEB_SERVER "192.168.5.115" #define WEB_PORT "1234" #define WEB_PATH "/"  
static const char *TAG = "example"; static const char *REQUEST = "GET " WEB_PATH  
" HTTP/1.0\r\n" "Host: " WEB_SERVER ":" WEB_PORT "\r\n" "User-Agent: esp-idf/1.0  
esp32\r\n" "\r\n"; uint8_t shtc3_read_temperature(uint16_t* temperature) { uint8_t
```

```
data[2] = {0x78, 0x66}; i2c_cmd_handle_t cmd = i2c_cmd_link_create();
i2c_master_start(cmd); i2c_master_write_byte(cmd, (SHTC3_ADDR << 1) |
I2C_MASTER_WRITE, true); i2c_master_write(cmd, data, 2, true);
i2c_master_stop(cmd); esp_err_t ret = i2c_master_cmd_begin(I2C_MASTER_NUM,
cmd, 1000 / portTICK_PERIOD_MS); i2c_cmd_link_delete(cmd); if (ret != ESP_OK) {
return ret; } vTaskDelay(20 / portTICK_PERIOD_MS); // Wait for the measurement to
complete uint8_t temperature_data[6]; cmd = i2c_cmd_link_create();
i2c_master_start(cmd); i2c_master_write_byte(cmd, (SHTC3_ADDR << 1) |
I2C_MASTER_READ, true); i2c_master_read(cmd, temperature_data, 6,
I2C_MASTER_LAST_NACK); i2c_master_stop(cmd); ret =
i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, 1000 / portTICK_PERIOD_MS);
i2c_cmd_link_delete(cmd); if (ret != ESP_OK) { return ret; } *temperature =
(temperature_data[0] << 8) | temperature_data[1]; return ESP_OK; } uint8_t
shtc3_read_humidity(uint16_t* humidity) { uint8_t data[2] = {0x78, 0x66};
i2c_cmd_handle_t cmd = i2c_cmd_link_create(); i2c_master_start(cmd);
i2c_master_write_byte(cmd, (SHTC3_ADDR << 1) | I2C_MASTER_WRITE, true);
i2c_master_write(cmd, data, 2, true); i2c_master_stop(cmd); esp_err_t ret =
i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, 1000 / portTICK_PERIOD_MS);
i2c_cmd_link_delete(cmd); if (ret != ESP_OK) { return ret; } vTaskDelay(20 /
portTICK_PERIOD_MS); // Wait for the measurement to complete uint8_t
humidity_data[6]; cmd = i2c_cmd_link_create(); i2c_master_start(cmd);
i2c_master_write_byte(cmd, (SHTC3_ADDR << 1) | I2C_MASTER_READ, true);
i2c_master_read(cmd, humidity_data, 6, I2C_MASTER_LAST_NACK);
i2c_master_stop(cmd); ret = i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, 1000 /
portTICK_PERIOD_MS); i2c_cmd_link_delete(cmd); if (ret != ESP_OK) { return ret; }
*humidity = (humidity_data[3] << 8) | humidity_data[4]; return ESP_OK; } static void
http_get_task(void *pvParameters) { const struct addrinfo hints = { .ai_family =
AF_INET, .ai_socktype = SOCK_STREAM, }; struct addrinfo *res; struct in_addr *addr;
int s, r; char recv_buf[64]; while(1) { int err = getaddrinfo(WEB_SERVER, WEB_PORT,
&hints, &res); if(err != 0 || res == NULL) { ESP_LOGE(TAG, "DNS lookup failed err=%d
res=%p", err, res); vTaskDelay(1000 / portTICK_PERIOD_MS); continue; } /* Code to
print the resolved IP. Note: inet_ntoa is non-reentrant, look at ipaddr_ntoa_r for "real"
code */ addr = &((struct sockaddr_in *)res->ai_addr)->sin_addr; ESP_LOGI(TAG, "DNS
lookup succeeded. IP=%s", inet_ntoa(*addr)); s = socket(res->ai_family,
res->ai_socktype, 0); if(s < 0) { ESP_LOGE(TAG, "... Failed to allocate socket.");
freeaddrinfo(res); vTaskDelay(1000 / portTICK_PERIOD_MS); continue; }
ESP_LOGI(TAG, "... allocated socket"); if(connect(s, res->ai_addr, res->ai_addrlen) !=
0) { ESP_LOGE(TAG, "... socket connect failed errno=%d", errno); close(s);
freeaddrinfo(res); vTaskDelay(4000 / portTICK_PERIOD_MS); continue; }
ESP_LOGI(TAG, "... connected"); freeaddrinfo(res); if (write(s, REQUEST,
```



```

strlen(REQUEST)) < 0) { ESP_LOGE(TAG, "... socket send failed"); close(s);
vTaskDelay(4000 / portTICK_PERIOD_MS); continue; } ESP_LOGI(TAG, "... socket
send success"); struct timeval receiving_timeout; receiving_timeout.tv_sec = 5;
receiving_timeout.tv_usec = 0; if (setsockopt(s, SOL_SOCKET, SO_RCVTIMEO,
&receiving_timeout, sizeof(receiving_timeout)) < 0) { ESP_LOGE(TAG, "... failed to set
socket receiving timeout"); close(s); vTaskDelay(4000 / portTICK_PERIOD_MS);
continue; } ESP_LOGI(TAG, "... set socket receiving timeout success"); /* Read HTTP
response */ do { bzero(recv_buf, sizeof(recv_buf)); r = read(s, recv_buf,
sizeof(recv_buf)-1); for(int i = 0; i < r; i++) { putchar(recv_buf[i]); } } while(r > 0);
ESP_LOGI(TAG, "... done reading from socket. Last read return=%d errno=%d.", r,
errno); close(s); for(int countdown = 10; countdown >= 0; countdown--) {
ESP_LOGI(TAG, "%d... ", countdown); vTaskDelay(1000 / portTICK_PERIOD_MS); }
ESP_LOGI(TAG, "Starting again!"); } } static void http_post_task(void *pvParameters) {
const struct addrinfo hints = { .ai_family = AF_INET, .ai_socktype = SOCK_STREAM, };
struct addrinfo *res; struct in_addr *addr; int s, r; char recv_buf[64]; while (1) { int err =
getaddrinfo(WEB_SERVER, WEB_PORT, &hints, &res); if (err != 0 || res == NULL) {
ESP_LOGE(TAG, "DNS lookup failed err=%d res=%p", err, res); vTaskDelay(1000 /
portTICK_PERIOD_MS); continue; } /* Code to print the resolved IP. Note: inet_ntoa is
non-reentrant, look at ipaddr_ntoa_r for "real" code */ addr = &((struct sockaddr_in
*)res->ai_addr)->sin_addr; ESP_LOGI(TAG, "DNS lookup succeeded. IP=%s",
inet_ntoa(*addr)); s = socket(res->ai_family, res->ai_socktype, 0); if (s < 0) {
ESP_LOGE(TAG, "... Failed to allocate socket."); freeaddrinfo(res); vTaskDelay(1000 /
portTICK_PERIOD_MS); continue; } ESP_LOGI(TAG, "... allocated socket"); if
(connect(s, res->ai_addr, res->ai_addrlen) != 0) { ESP_LOGE(TAG, "... socket connect
failed errno=%d", errno); close(s); freeaddrinfo(res); vTaskDelay(4000 /
portTICK_PERIOD_MS); continue; } ESP_LOGI(TAG, "... connected");
freeaddrinfo(res); uint16_t temperature, humidity;
shtc3_read_temperature(&temperature); shtc3_read_humidity(&humidity); float
temperature_celsius = ((.45+175*(temperature / 65535.0)) - 32) * (0.56); float
humidity_percentage = 100* (humidity / 65535.0); printf("Temperature is %.2fC with a
%.2f%% humidity\n", temperature_celsius, humidity_percentage);
vTaskDelay(pdMS_TO_TICKS(2000)); // 2-second delay // Modified request for HTTP
POST const char *post_request = "POST " WEB_PATH " HTTP/1.0\r\n" "Host:
"WEB_SERVER":"WEB_PORT"\r\n" "Content-Type:
application/x-www-form-urlencoded\r\n" "Content-Length: %d\r\n\r\n"
"temperature=%.2f&humidity=%.2f"; char request_buffer[128]; snprintf(request_buffer,
sizeof(request_buffer), post_request, (int)(strlen("temperature=&humidity=") + 2 *
sizeof(float)), temperature_celsius, humidity_percentage); if (write(s, request_buffer,
strlen(request_buffer)) < 0) { ESP_LOGE(TAG, "... socket send failed"); close(s);
vTaskDelay(4000 / portTICK_PERIOD_MS); continue; } ESP_LOGI(TAG, "... socket

```

```
send success"); struct timeval receiving_timeout; receiving_timeout.tv_sec = 5;
receiving_timeout.tv_usec = 0; if (setsockopt(s, SOL_SOCKET, SO_RCVTIMEO,
&receiving_timeout, sizeof(receiving_timeout)) < 0) { ESP_LOGE(TAG, "... failed to set
socket receiving timeout"); close(s); vTaskDelay(4000 / portTICK_PERIOD_MS);
continue; } ESP_LOGI(TAG, "... set socket receiving timeout success"); /* Read HTTP
response */ do { bzero(recv_buf, sizeof(recv_buf)); r = read(s, recv_buf, sizeof(recv_buf)
- 1); for (int i = 0; i < r; i++) { putchar(recv_buf[i]); } } while (r > 0); ESP_LOGI(TAG, "...
done reading from socket. Last read return=%d errno=%d.", r, errno); close(s); for (int
countdown = 10; countdown >= 0; countdown--) { ESP_LOGI(TAG, "%d... ",
countdown); vTaskDelay(1000 / portTICK_PERIOD_MS); } ESP_LOGI(TAG, "Starting
again!"); } } void starti2c(){ i2c_config_t conf = { .mode = I2C_MODE_MASTER,
.sda_io_num = I2C_MASTER_SDA_IO, .sda_pullup_en = GPIO_PULLUP_ENABLE,
.scl_io_num = I2C_MASTER_SCL_IO, .scl_pullup_en = GPIO_PULLUP_ENABLE,
.master.clk_speed = I2C_MASTER_FREQ_HZ, };
i2c_param_config(I2C_MASTER_NUM, &conf); i2c_driver_install(I2C_MASTER_NUM,
conf.mode, 0, 0, 0); } void app_main(void) { starti2c(); ESP_ERROR_CHECK(
nvs_flash_init() ); ESP_ERROR_CHECK(esp_netif_init());
ESP_ERROR_CHECK(esp_event_loop_create_default()); /* This helper function
configures Wi-Fi or Ethernet, as selected in menuconfig. * Read "Establishing Wi-Fi or
Ethernet Connection" section in * examples/protocols/README.md for more information
about this function. */ ESP_ERROR_CHECK(example_connect()); while (1) {
xTaskCreate(&http_post_task, "http_post_task", 4096, NULL, 5, NULL); } }
```

ChatGPT

Your code looks good for performing an HTTP POST request to the specified server and port with temperature and humidity data. I do have a couple of suggestions for improvement and clarification:

1. **Delay in the Main Loop:** You have a `vTaskDelay(pdMS_TO_TICKS(2000))`; in the main loop of `http_post_task`. Note that this delay is independent of the HTTP request interval. If you want to control how often the HTTP POST request is sent, you might want to consider moving this delay outside the `while(1)` loop in `http_post_task` and place it inside the main loop in `app_main` after the `xTaskCreate` call. This ensures that the delay is applied after each HTTP POST request.

c

```
while (1) {
    xTaskCreate(&http_post_task, "http_post_task", 4096, NULL,
5, NULL);
```

```
    vTaskDelay(pdMS_TO_TICKS(2000)); // 2-second delay before
the next HTTP POST
}
```

I2C Initialization: You might want to add some error checking and handling for the I2C initialization in the `starti2c` function. If there's an issue with I2C initialization, it's good to have error messages or actions to handle potential problems.

C

```
void starti2c() {
    i2c_config_t conf = {
        .mode = I2C_MODE_MASTER,
        .sda_io_num = I2C_MASTER_SDA_IO,
        .sda_pullup_en = GPIO_PULLUP_ENABLE,
        .scl_io_num = I2C_MASTER_SCL_IO,
        .scl_pullup_en = GPIO_PULLUP_ENABLE,
        .master.clk_speed = I2C_MASTER_FREQ_HZ,
    };

    esp_err_t i2c_err = i2c_param_config(I2C_MASTER_NUM, &conf);
    if (i2c_err != ESP_OK) {
        ESP_LOGE(TAG, "I2C parameter config error: %s",
esp_err_to_name(i2c_err));
        // Handle the error, e.g., abort the program or take
corrective actions.
        // ...
    }

    i2c_err = i2c_driver_install(I2C_MASTER_NUM, conf.mode, 0,
0, 0);
    if (i2c_err != ESP_OK) {
        ESP_LOGE(TAG, "I2C driver install error: %s",
esp_err_to_name(i2c_err));
        // Handle the error, e.g., abort the program or take
corrective actions.
        // ...
    }
}
```

- 1.
2. **Error Handling in HTTP POST Task:** It might be beneficial to add more robust error handling in the `http_post_task` function, especially for the HTTP POST request. If the request fails, it's good to log an error message or take appropriate actions.
3. **Memory Management:** Ensure that your tasks have enough stack space. In your `xTaskCreate` calls, you have allocated 4096 bytes for the stack (4096 as the third parameter). Ensure that this is sufficient for your tasks' requirements.

With these considerations, your code should work well for sending periodic HTTP POST requests with temperature and humidity data.



```
/home/ubuntu/esp/lab6/http_request/main/http_request_example_main.c:255:36: error:
'%.2f' directive output truncated writing between 3 and 313 bytes into a region of size 2
[-Werror=format-truncation=] 255 | const char *post_request = "POST " WEB_PATH "
HTTP/1.0\r\n" | ^~~~~~
```

ChatGPT

The error you're encountering is related to the format specifier in the `snprintf` function, specifically the "%.2f" specifier. This error indicates that the formatted output is being truncated because the target buffer (`request_buffer` in this case) is not large enough to accommodate the formatted string.

To fix this issue, you should ensure that the `request_buffer` has enough space to hold the formatted string. In your case, you are currently using a buffer size of 128, which may not be sufficient.

Increase the size of `request_buffer` to a value that can accommodate the entire formatted string. For safety, you might want to make it large enough to handle worst-case scenarios. For example:

```
c
char request_buffer[512]; // Increase the buffer size as needed
snprintf(request_buffer, sizeof(request_buffer), post_request,
          (int)(strlen("temperature=&humidity=") + 2 *
sizeof(float)),
          temperature_celsius, humidity_percentage);
```

This way, you ensure that the buffer has enough space for the formatted string, and you avoid the "format-truncation" error. Adjust the size according to the maximum length of the formatted string you expect to generate.