PROJECT : PID Temperature Control System (RTOS)

FOLDER STRUCTURE & SOURCE FILES

PROJECT FOLDER STRUCTURE

```
pid_temp_controller/
├── CMakeLists.txt
├── sdkconfig.defaults
└── main
    ├── CMakeLists.txt
    ├── main.c
    ├── dht11.c
    ├── dht11.h
    ├── lcd.c
    ├── lcd.h
    ├── pid.c
    ├── pid.h
    └── relay.h
```

ROOT CMakeLists.txt

```
cmake_minimum_required(VERSION 3.5)
include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(pid_temp_controller)
```

---

main/CMakeLists.txt

```
idf_component_register(
    SRCS "main.c" "dht11.c" "lcd.c" "pid.c"
    INCLUDE_DIRS "."
)
```

---

main/relay.h (GPIO pin mapping + macros)

```
#pragma once

#define HEATER_GPIO     18
#define FAN_GPIO        19
```

---

main/pid.h

```
#pragma once

typedef struct {
    float Kp, Ki, Kd;
    float prev_error;
    float integral;
} pid_t;

void pid_init(pid_t *pid, float Kp, float Ki, float Kd);
float pid_compute(pid_t *pid, float setpoint, float measured, float dt);
```

---

main/pid.c

```
#include "pid.h"

void pid_init(pid_t *pid, float Kp, float Ki, float Kd) {
    pid->Kp = Kp;
    pid->Ki = Ki;
    pid->Kd = Kd;
    pid->prev_error = 0;
    pid->integral = 0;
}

float pid_compute(pid_t *pid, float setpoint, float measured, float dt) {
    float error = setpoint - measured;
    pid->integral += error * dt;
    float derivative = (error - pid->prev_error) / dt;
    pid->prev_error = error;

    float output = pid->Kp * error + pid->Ki * pid->integral + pid->Kd *
derivative;

    if(output > 100) output = 100;
    if(output < 0) output = 0;

    return output;
}
```

---

main/dht11.h

```
#pragma once
#include "esp_err.h"

esp_err_t dht11_init(int gpio_num);
esp_err_t dht11_read(float *temperature, float *humidity);
```

---

main/dht11.c

```
#include "dht11.h"
#include "driver/gpio.h"
```

```
#include "esp_timer.h"
#include "esp_log.h"

static const char *TAG = "DHT11";
static int dht_gpio;

esp_err_t dht11_init(int gpio_num) {
    dht_gpio = gpio_num;
    gpio_set_direction(dht_gpio, GPIO_MODE_OUTPUT);
    gpio_set_level(dht_gpio, 1);
    return ESP_OK;
}

static int wait_for_level(int level, uint32_t timeout_us) {
    int64_t start = esp_timer_get_time();
    while(gpio_get_level(dht_gpio) == level) {
        if((esp_timer_get_time() - start) > timeout_us) return -1;
    }
    return 0;
}

esp_err_t dht11_read(float *temperature, float *humidity) {
    uint8_t data[5] = {0};

    gpio_set_direction(dht_gpio, GPIO_MODE_OUTPUT);
    gpio_set_level(dht_gpio, 0);
    ets_delay_us(20000);

    gpio_set_level(dht_gpio, 1);
    ets_delay_us(40);
    gpio_set_direction(dht_gpio, GPIO_MODE_INPUT);

    if(wait_for_level(1, 80) < 0) return ESP_FAIL;
    if(wait_for_level(0, 80) < 0) return ESP_FAIL;

    for(int i = 0; i < 40; i++) {
        if(wait_for_level(1, 50) < 0) return ESP_FAIL;
        int64_t t = esp_timer_get_time();
        if(wait_for_level(0, 70) < 0) return ESP_FAIL;
        int64_t dt = esp_timer_get_time() - t;
        data[i/8] <<= 1;
        if(dt > 40) data[i/8] |= 1;
    }

    uint8_t checksum = data[0]+data[1]+data[2]+data[3];
    if(checksum != data[4]) return ESP_FAIL;

    *humidity = data[0];
    *temperature = data[2];

    return ESP_OK;
}
```

---

main/lcd.h

```
#pragma once
#include <stdint.h>

void lcd_init(void);
```

```
void lcd_clear(void);
void lcd_putc(char c);
void lcd_puts(uint8_t col, uint8_t row, const char *s);
```

---

main/lcd.c (4-bit mode driver)

```
#include "lcd.h"
#include "driver/gpio.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

#define LCD_RS 14
#define LCD_EN 27
#define LCD_D4 26
#define LCD_D5 25
#define LCD_D6 33
#define LCD_D7 32

static void lcd_send_nibble(uint8_t nib) {
    gpio_set_level(LCD_D4, (nib >> 0) & 0x01);
    gpio_set_level(LCD_D5, (nib >> 1) & 0x01);
    gpio_set_level(LCD_D6, (nib >> 2) & 0x01);
    gpio_set_level(LCD_D7, (nib >> 3) & 0x01);

    gpio_set_level(LCD_EN, 1);
    ets_delay_us(1);
    gpio_set_level(LCD_EN, 0);
    ets_delay_us(100);
}

static void lcd_cmd(uint8_t cmd) {
    gpio_set_level(LCD_RS, 0);
    lcd_send_nibble(cmd >> 4);
    lcd_send_nibble(cmd & 0x0F);
    vTaskDelay(pdMS_TO_TICKS(2));
}

static void lcd_data(uint8_t data) {
    gpio_set_level(LCD_RS, 1);
    lcd_send_nibble(data >> 4);
    lcd_send_nibble(data & 0x0F);
}

void lcd_init(void) {
    gpio_set_direction(LCD_RS, GPIO_MODE_OUTPUT);
    gpio_set_direction(LCD_EN, GPIO_MODE_OUTPUT);
    gpio_set_direction(LCD_D4, GPIO_MODE_OUTPUT);
    gpio_set_direction(LCD_D5, GPIO_MODE_OUTPUT);
    gpio_set_direction(LCD_D6, GPIO_MODE_OUTPUT);
    gpio_set_direction(LCD_D7, GPIO_MODE_OUTPUT);

    vTaskDelay(pdMS_TO_TICKS(50));

    lcd_send_nibble(0x03);
    vTaskDelay(pdMS_TO_TICKS(5));
    lcd_send_nibble(0x03);
    vTaskDelay(pdMS_TO_TICKS(5));
    lcd_send_nibble(0x02);
```

```
    lcd_cmd(0x28);
    lcd_cmd(0x0C);
    lcd_cmd(0x06);
    lcd_cmd(0x01);
}

void lcd_clear(void) {
    lcd_cmd(0x01);
}

void lcd_putc(char c) {
    lcd_data(c);
}

void lcd_puts(uint8_t col, uint8_t row, const char *s) {
    uint8_t addr = (row == 0) ? (0x80 + col) : (0xC0 + col);
    lcd_cmd(addr);
    while(*s) lcd_putc(*s++);
}
```

---

main/main.c (FreeRTOS tasks + PID + control)

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/semphr.h"

#include "driver/gpio.h"
#include "esp_log.h"

#include "dht11.h"
#include "lcd.h"
#include "pid.h"
#include "relay.h"

static const char *TAG = "MAIN";

typedef struct {
    float temp;
    float hum;
} sensor_t;

static sensor_t g_sensor = {0};
static SemaphoreHandle_t xMutex;

static float g_setpoint = 40.0;
static float g_pid_output = 0.0;

void task_dht(void *pv) {
    while(1) {
        float t, h;
        if(dht11_read(&t, &h) == ESP_OK) {
            xSemaphoreTake(xMutex, portMAX_DELAY);
            g_sensor.temp = t;
            g_sensor.hum = h;
            xSemaphoreGive(xMutex);
        }
        vTaskDelay(pdMS_TO_TICKS(2000));
    }
```

```
}

void task_pid(void *pv) {
    pid_t pid;
    pid_init(&pid, 3.0, 0.5, 1.0);
    TickType_t last = xTaskGetTickCount();

    while(1) {
        xSemaphoreTake(xMutex, portMAX_DELAY);
        float t = g_sensor.temp;
        xSemaphoreGive(xMutex);

        g_pid_output = pid_compute(&pid, g_setpoint, t, 1.0);
        vTaskDelayUntil(&last, pdMS_TO_TICKS(1000));
    }
}

void task_heater(void *pv) {
    const int window_ms = 5000;

    while(1) {
        float on_ms = (g_pid_output/100.0f) * window_ms;
        TickType_t start = xTaskGetTickCount();

        if(on_ms > 0) gpio_set_level(HEATER_GPIO, 1);

        while((xTaskGetTickCount() - start) < pdMS_TO_TICKS(on_ms)) {
            vTaskDelay(pdMS_TO_TICKS(100));
        }

        gpio_set_level(HEATER_GPIO, 0);

        while((xTaskGetTickCount() - start) < pdMS_TO_TICKS(window_ms)) {
            vTaskDelay(pdMS_TO_TICKS(100));
        }
    }
}

void task_fan(void *pv) {
    while(1) {
        xSemaphoreTake(xMutex, portMAX_DELAY);
        float t = g_sensor.temp;
        xSemaphoreGive(xMutex);

        if(t > g_setpoint + 2) gpio_set_level(FAN_GPIO, 1);
        else gpio_set_level(FAN_GPIO, 0);

        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}

void task_lcd(void *pv) {
    char buf[16];
    while(1) {
        xSemaphoreTake(xMutex, portMAX_DELAY);
        float t = g_sensor.temp;
        float h = g_sensor.hum;
        xSemaphoreGive(xMutex);

        lcd_clear();
        snprintf(buf, 16, "T:%.1f SP:%.0f", t, g_setpoint);
```

```
        lcd_puts(0,0, buf);

        snprintf(buf, 16, "OUT:%3.0f%%", g_pid_output);
        lcd_puts(0,1, buf);

        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}

void app_main(void) {
    xMutex = xSemaphoreCreateMutex();

    gpio_set_direction(HEATER_GPIO, GPIO_MODE_OUTPUT);
    gpio_set_direction(FAN_GPIO, GPIO_MODE_OUTPUT);

    dht11_init(4);
    lcd_init();

    xTaskCreate(task_dht, "task_dht", 2048, NULL, 4, NULL);
    xTaskCreate(task_pid, "task_pid", 2048, NULL, 5, NULL);
    xTaskCreate(task_heater, "task_heater", 2048, NULL, 3, NULL);
    xTaskCreate(task_fan, "task_fan", 2048, NULL, 2, NULL);
    xTaskCreate(task_lcd, "task_lcd", 2048, NULL, 1, NULL);
}
```
