IoT lab Led relay

C20452282

Program to light up led

#include <wiringPi.h>

#include <softPwm.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

#define LED\_PIN 9

volatile int running = 1;

// Signal handler to gracefully exit the program

void sigintHandler(int signal) {

running = 0;

}

// Function to generate a sine wave pattern on the LED

void sineWaveLED() {

double frequency = 0.5; // Adjust this value to change the speed of the sine wave

while (running) {

for (int i = 0; i < 360; i += 10) {

// Calculate the sine value for the angle in radians

double sineValue = sin((double)i \* M\_PI / 180.0);

// Map the sine value to PWM duty cycle (0 to 100)

int dutyCycle = (int)((sineValue + 1) \* 50);

// Set PWM duty cycle to control LED brightness

softPwmWrite(LED\_PIN, dutyCycle);

delay(10); // Adjust delay time for speed

}

}

}

int main() {

// Set up WiringPi and soft PWM

if (wiringPiSetup() == -1) {

fprintf(stderr, "Unable to setup wiringPi. Exiting.\n");

return 1;

}

// Set up signal handler for graceful exit

signal(SIGINT, sigintHandler);

// Set up soft PWM for the LED

softPwmCreate(LED\_PIN, 0, 100); // Initialize soft PWM with 0% duty cycle

// Run the sine wave pattern

sineWaveLED();

// Clean up

softPwmWrite(LED\_PIN, 0); // Turn off the LED

return 0;

}

Extend your programme to accept a message from a neighbouring Raspberry Pi, and make this initiate the LED

#include <wiringPi.h>

#include <softPwm.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define LED\_PIN 9

#define PORT 8888

volatile int running = 1;

// Signal handler to gracefully exit the program

void sigintHandler(int signal) {

running = 0;

}

// Function to generate a sine wave pattern on the LED

void sineWaveLED() {

double frequency = 0.5; // Adjust this value to change the speed of the sine wave

while (running) {

for (int i = 0; i < 360; i += 10) {

// Calculate the sine value for the angle in radians

double sineValue = sin((double)i \* M\_PI / 180.0);

// Map the sine value to PWM duty cycle (0 to 100)

int dutyCycle = (int)((sineValue + 1) \* 50);

// Set PWM duty cycle to control LED brightness

softPwmWrite(LED\_PIN, dutyCycle);

delay(10); // Adjust delay time for speed

}

}

}

// Function to set up a simple socket server

void initSocketServer() {

int serverSocket, newSocket;

struct sockaddr\_in serverAddr, clientAddr;

socklen\_t addrSize = sizeof(struct sockaddr\_in);

char buffer[1024] = {0};

// Create socket

if ((serverSocket = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set up server information

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_addr.s\_addr = INADDR\_ANY;

serverAddr.sin\_port = htons(PORT);

// Bind the socket to the specified port

if (bind(serverSocket, (struct sockaddr \*)&serverAddr, sizeof(serverAddr)) < 0) {

perror("Bind failed");

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(serverSocket, 3) < 0) {

perror("Listen failed");

exit(EXIT\_FAILURE);

}

// Accept incoming connection

if ((newSocket = accept(serverSocket, (struct sockaddr \*)&clientAddr, &addrSize)) < 0) {

perror("Accept failed");

exit(EXIT\_FAILURE);

}

// Receive message from the client

while (recv(newSocket, buffer, sizeof(buffer), 0) > 0) {

printf("Received message: %s\n", buffer);

printf("Initiating LED...\n");

sineWaveLED();

}

}

int main() {

// Set up WiringPi and soft PWM

if (wiringPiSetup() == -1) {

fprintf(stderr, "Unable to setup wiringPi. Exiting.\n");

return 1;

}

// Set up signal handler for graceful exit

signal(SIGINT, sigintHandler);

// Set up soft PWM for the LED

softPwmCreate(LED\_PIN, 0, 100); // Initialize soft PWM with 0% duty cycle

// Initialize socket server

initSocketServer();

// Clean up

softPwmWrite(LED\_PIN, 0); // Turn off the LED

return 0;

}

Q3

#include <wiringPi.h>

#include <softPwm.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define LED\_PIN 9

#define PORT 8888 // Port for communication

#define RIGHT\_NEIGHBOR\_IP "192.168.1.2"

#define RIGHT\_NEIGHBOR\_PORT 8888

volatile int running = 1;

// Signal handler to gracefully exit the program

void sigintHandler(int signal) {

running = 0;

}

// Function to generate a sine wave pattern on the LED

void sineWaveLED() {

double frequency = 0.5; // Adjust this value to change the speed of the sine wave

while (running) {

for (int i = 0; i < 360; i += 10) {

// Calculate the sine value for the angle in radians

double sineValue = sin((double)i \* M\_PI / 180.0);

// Map the sine value to PWM duty cycle (0 to 100)

int dutyCycle = (int)((sineValue + 1) \* 50);

// Set PWM duty cycle to control LED brightness

softPwmWrite(LED\_PIN, dutyCycle);

delay(10); // Adjust delay time for speed

}

}

}

// Function to set up a simple socket server

void initSocketServer() {

int serverSocket, newSocket;

struct sockaddr\_in serverAddr, clientAddr;

socklen\_t addrSize = sizeof(struct sockaddr\_in);

char buffer[1024] = {0};

// Create socket

if ((serverSocket = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set up server information

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_addr.s\_addr = INADDR\_ANY;

serverAddr.sin\_port = htons(PORT);

// Bind the socket to the specified port

if (bind(serverSocket, (struct sockaddr \*)&serverAddr, sizeof(serverAddr)) < 0) {

perror("Bind failed");

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(serverSocket, 3) < 0) {

perror("Listen failed");

exit(EXIT\_FAILURE);

}

// Accept incoming connection

if ((newSocket = accept(serverSocket, (struct sockaddr \*)&clientAddr, &addrSize)) < 0) {

perror("Accept failed");

exit(EXIT\_FAILURE);

}

// Receive message from the client

while (recv(newSocket, buffer, sizeof(buffer), 0) > 0) {

printf("Received message: %s\n", buffer);

// You can add logic here to initiate the LED based on the received message

// For simplicity, let's just initiate the LED when any message is received

printf("Initiating LED...\n");

sineWaveLED();

// Send a message to the right neighbor

int clientSocket;

struct sockaddr\_in clientAddr;

// Create socket

if ((clientSocket = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set up client information

clientAddr.sin\_family = AF\_INET;

clientAddr.sin\_addr.s\_addr = inet\_addr(RIGHT\_NEIGHBOR\_IP);

clientAddr.sin\_port = htons(RIGHT\_NEIGHBOR\_PORT);

// Connect to the right neighbor

if (connect(clientSocket, (struct sockaddr \*)&clientAddr, sizeof(clientAddr)) < 0) {

perror("Connection to right neighbor failed");

exit(EXIT\_FAILURE);

}

// Send a message to the right neighbor

const char \*message = "LED initiated from the left!";

send(clientSocket, message, strlen(message), 0);

close(clientSocket);

}

}

int main() {

// Set up WiringPi and soft PWM

if (wiringPiSetup() == -1) {

fprintf(stderr, "Unable to setup wiringPi. Exiting.\n");

return 1;

}

// Set up signal handler for graceful exit

signal(SIGINT, sigintHandler);

// Set up soft PWM for the LED

softPwmCreate(LED\_PIN, 0, 100); // Initialize soft PWM with 0% duty cycle

// Initialize socket server

initSocketServer();

// Clean up

softPwmWrite(LED\_PIN, 0); // Turn off the LED

return 0;

}

Q4

#include <wiringPi.h>

#include <softPwm.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define LED\_PIN 9

#define PORT 8888 // Port for communication

#define RIGHT\_NEIGHBOR\_IP "192.168.1.2"

#define RIGHT\_NEIGHBOR\_PORT 8888

#define LEFT\_NEIGHBOR\_IP "192.168.1.3"

#define LEFT\_NEIGHBOR\_PORT 8888

volatile int running = 1;

// Signal handler to gracefully exit the program

void sigintHandler(int signal) {

running = 0;

}

// Function to generate a sine wave pattern on the LED

void sineWaveLED() {

double frequency = 0.5; // Adjust this value to change the speed of the sine wave

while (running) {

for (int i = 0; i < 360; i += 10) {

// Calculate the sine value for the angle in radians

double sineValue = sin((double)i \* M\_PI / 180.0);

// Map the sine value to PWM duty cycle (0 to 100)

int dutyCycle = (int)((sineValue + 1) \* 50);

// Set PWM duty cycle to control LED brightness

softPwmWrite(LED\_PIN, dutyCycle);

delay(10); // Adjust delay time for speed

}

}

}

// Function to send a message to the specified IP and port

void sendMessage(const char \*ip, int port, const char \*message) {

int clientSocket;

struct sockaddr\_in clientAddr;

// Create socket

if ((clientSocket = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set up client information

clientAddr.sin\_family = AF\_INET;

clientAddr.sin\_addr.s\_addr = inet\_addr(ip);

clientAddr.sin\_port = htons(port);

// Connect to IP

if (connect(clientSocket, (struct sockaddr \*)&clientAddr, sizeof(clientAddr)) < 0) {

perror("Connection failed");

exit(EXIT\_FAILURE);

}

// Send the message

send(clientSocket, message, strlen(message), 0);

// Close the socket

close(clientSocket);

}

int main() {

// Set up WiringPi and soft PWM

if (wiringPiSetup() == -1) {

fprintf(stderr, "Unable to setup wiringPi. Exiting.\n");

return 1;

}

// Set up signal handler for graceful exit

signal(SIGINT, sigintHandler);

// Set up soft PWM for the LED

softPwmCreate(LED\_PIN, 0, 100); // Initialize soft PWM with 0% duty cycle

// Initialize socket server

initSocketServer();

// Run the sine wave pattern

sineWaveLED();

// After completing the LED cycle, send a message to the left neighbor

const char \*leftNeighborMessage = "LED cycle completed!";

sendMessage(LEFT\_NEIGHBOR\_IP, LEFT\_NEIGHBOR\_PORT, leftNeighborMessage);

// Clean up

softPwmWrite(LED\_PIN, 0); // Turn off the LED

return 0;

}