Plug-and-Play GPIO Control Device

Introduction:

The Raspberry Pi GPIO Control Device is a Python-based framework developed by Krishna Ganta, a student of Northshore School District (Student ID: 2032673). It is a script to run on the Raspberry Pi family of microcontrollers, that allows manual control of the GPIO pins remotely via a TCP connection. Note that to function, either a wireless adapter OR the W family of Raspberry Pi is required. If these are not available, consider using the Arduino GPIO Control Device, which uses a USB serial connection.

Purpose:

The purpose of this Python script is to create a plug-and-play device that acts as a placeholder for future circuits. It allows manual control of GPIO (General Purpose Input/Output) pins on a Raspberry Pi. This device is designed to facilitate the manual control of GPIO pins until they are automated in the future.

Libraries Used:

1. **socket**: This library is used for socket communication, allowing the Raspberry Pi to receive commands from a remote controller.
2. **RPi.GPIO**: This library provides functionality for controlling GPIO pins on the Raspberry Pi.

GPIO Pin Configuration:

- The `GPIO.setmode(GPIO.BCM)` line sets the GPIO pin numbering mode to Broadcom SOC channel numbers.
- The `gpio_pins` list contains the GPIO pins that will be controlled by the device. You should replace these with the actual GPIO pins that correspond to your future automated circuits.

GPIO Pin Initialization:

- The script sets up the GPIO pins defined in the `gpio_pins` list as outputs and initializes them to a low (OFF) state. This is done to ensure that the pins start in a known state.

Socket Server Configuration:

- The `host` variable should be set to the IP address of the server where commands will be sent from. This is the device that will control the Raspberry Pi.
- The `port` variable should be set to the desired port number for socket communication.

Handling Incoming Messages:

- The `handle_message` function is responsible for processing incoming messages from the controller.
- It expects messages in the format "`COMMAND PIN_NUMBER`," where "COMMAND" can be either "ON" or "OFF," and "PIN_NUMBER" is the GPIO pin number to be controlled.

Message Handling Logic:

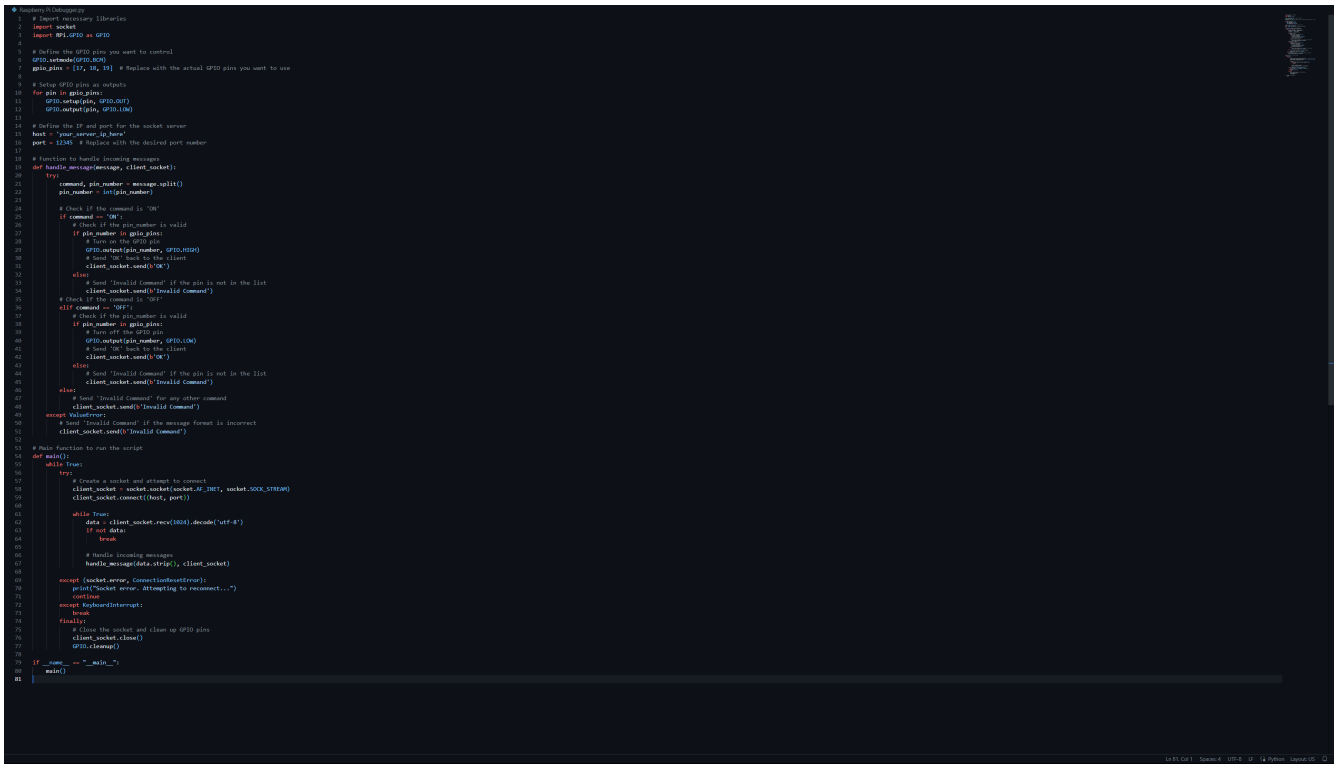
- The script checks if the received command is "ON" or "OFF." If it's "ON," it turns on the specified GPIO pin, and if it's "OFF," it turns off the pin.
- It verifies that the specified `PIN_NUMBER` is in the list of allowed GPIO pins (`gpio_pins`). If the pin is valid, the script performs the requested action; otherwise, it responds with "Invalid Command."
- If the message format is incorrect (e.g., missing or invalid `PIN_NUMBER`), it responds with "Invalid Command."

Main Loop:

- The `main` function contains the main execution loop of the script.
- It continuously attempts to create a socket connection to the controller (defined by `host` and `port`).
- Once connected, it listens for incoming messages and passes them to the `handle_message` function for processing.
- If there is a socket error or a connection reset, it attempts to reconnect to the controller.
- The script can be terminated by a keyboard interrupt, or (*nix only) sending a SIGINT to the process.

Cleanup:

- The script closes the socket and cleans up GPIO pins (restores them to their original state) when it exits.


Raspberry Pi Debugging.py
# Import necessary libraries
import socket
import RPi.GPIO as GPIO
# Define the GPIO pins you want to control
GPIO.setmode(GPIO.BCM)
gpio_pins = [17, 18, 19] # Replace with the actual GPIO pins you want to use
# Setup GPIO pins as outputs
for pin in gpio_pins:
GPIO.setup(pin, GPIO.OUT)
GPIO.output(pin, GPIO.LOW)
# Define the IP and port for the socket server
host = 'your_server_ip_here'
port = 12345 # Replace with the desired port number
# Function to handle incoming messages
def handle_message(message, client_socket):
try:
command, pin_number = message.split()
pin_number = int(pin_number)
# Check if the command is 'ON'
if command == 'ON':
# Check if the pin_number is valid
if pin_number in gpio_pins:
# Turn on the GPIO pin
GPIO.output(pin_number, GPIO.HIGH)
# Send 'OK' back to the client
client_socket.send(b'OK')
else:
# Send 'Invalid Command' if the pin is not in the list
client_socket.send(b'Invalid Command')
# Check if the command is 'OFF'
elif command == 'OFF':
# Check if the pin_number is valid
if pin_number in gpio_pins:
# Turn off the GPIO pin
GPIO.output(pin_number, GPIO.LOW)
# Send 'OK' back to the client
client_socket.send(b'OK')
else:
# Send 'Invalid Command' if the pin is not in the list
client_socket.send(b'Invalid Command')
else:
# Send 'Invalid Command' for any other command
client_socket.send(b'Invalid Command')
except ValueError:
# Send 'Invalid Command' if the message format is incorrect
client_socket.send(b'Invalid Command')
# Main function to run the script
def main():
while True:
try:
# Create a socket and attempt to connect
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((host, port))
while True:
data = client_socket.recv(1024).decode('utf-8')
if not data:
break
# Handle incoming messages
handle_message(data.strip(), client_socket)
except (socket.error, ConnectionResetError):
print("Socket error. Attempting to reconnect...")
continue
except KeyboardInterrupt:
break
finally:
# Close the socket and clean up GPIO pins
client_socket.close()
GPIO.cleanup()
if __name__ == "__main__":
main()
Ln 81, Col 1 Spaces: 4 UTF-8 LF Python Layout: US