import RPi.GPIO as GPIO

from mfrc522 import SimpleMFRC522

import LCD1602

import threading

import time

# Define pin numbers for gas sensor

GAS\_SENSOR\_PIN = 17

EXHAUST\_FAN\_PIN = 0

BUZZER\_PIN = 24

# Define pin numbers for RFID

RFID\_READ\_LED\_PIN = 19

SOLENOID\_LOCK\_PIN = 27

ACCESS\_GRANTED\_PIN = 26

ACCESS\_DENIED\_PIN = 13

# Define pin numbers for fire sensor

FIRE\_SENSOR\_PIN = 21

POWER\_SUPPLY\_PIN = 22

EMERGENCY\_LIGHT\_PIN = 23

WATER\_PUMP\_PIN = 5

# Setup GPIO for gas sensor

GPIO.setmode(GPIO.BCM)

GPIO.setup(GAS\_SENSOR\_PIN, GPIO.IN)

GPIO.setup(EXHAUST\_FAN\_PIN, GPIO.OUT)

GPIO.setup(BUZZER\_PIN, GPIO.OUT)

# Setup GPIO for RFID

GPIO.setup(RFID\_READ\_LED\_PIN, GPIO.OUT)

GPIO.setup(SOLENOID\_LOCK\_PIN, GPIO.OUT)

GPIO.setup(ACCESS\_GRANTED\_PIN, GPIO.OUT)

GPIO.setup(ACCESS\_DENIED\_PIN, GPIO.OUT)

# Setup GPIO for fire sensor

GPIO.setup(FIRE\_SENSOR\_PIN, GPIO.IN)

GPIO.setup(POWER\_SUPPLY\_PIN, GPIO.OUT)

GPIO.setup(EMERGENCY\_LIGHT\_PIN, GPIO.OUT)

GPIO.setup(WATER\_PUMP\_PIN, GPIO.OUT)

# Initialize the output pins to their default states for fire sensor

GPIO.output(POWER\_SUPPLY\_PIN, GPIO.HIGH) # Inverted logic: HIGH means off

GPIO.output(EMERGENCY\_LIGHT\_PIN, GPIO.LOW) # Normal logic

GPIO.output(WATER\_PUMP\_PIN, GPIO.HIGH) # Inverted logic: HIGH means off

# Initialize RFID reader and LCD

reader = SimpleMFRC522()

try:

LCD1602.init(0x27, 1) # Change address to 0x27 or 0x3F based on your setup

print("LCD initialized successfully")

except Exception as e:

print("Error initializing LCD:", e)

# Authorized RFID tags

AUTHORIZED\_IDS = [425546492625] # Replace with your authorized RFID IDs

def clear\_lcd():

LCD1602.write(0, 0, " ")

LCD1602.write(1, 0, " ")

def display\_message(line1, line2, delay=2):

try:

LCD1602.write(0, 0, line1)

LCD1602.write(1, 0, line2)

print("Message displayed on LCD:", line1, line2)

time.sleep(delay)

clear\_lcd()

except Exception as e:

print("Error displaying message on LCD:", e)

# Function to monitor gas sensor

def gas\_sensor\_monitor():

try:

while True:

gas\_detected = not GPIO.input(GAS\_SENSOR\_PIN) # Invert logic if needed

if gas\_detected:

GPIO.output(EXHAUST\_FAN\_PIN, GPIO.LOW) # Turn on exhaust fan

GPIO.output(BUZZER\_PIN, GPIO.HIGH) # Turn on buzzer

display\_message("Gas Detected", "", delay=3)

print("Gas Detected")

else:

GPIO.output(EXHAUST\_FAN\_PIN, GPIO.HIGH) # Turn off exhaust fan

GPIO.output(BUZZER\_PIN, GPIO.LOW) # Turn off buzzer

time.sleep(0.5) # Adjust sleep time as needed

except KeyboardInterrupt:

pass

# Function to monitor RFID

def rfid\_sensor\_monitor(reader):

try:

while True:

GPIO.output(RFID\_READ\_LED\_PIN, GPIO.HIGH)

print("Hold a tag near the reader")

id, text = reader.read()

GPIO.output(RFID\_READ\_LED\_PIN, GPIO.LOW)

if id in AUTHORIZED\_IDS:

GPIO.output(SOLENOID\_LOCK\_PIN, GPIO.LOW) # Open the lock

GPIO.output(ACCESS\_GRANTED\_PIN, GPIO.HIGH)

GPIO.output(ACCESS\_DENIED\_PIN, GPIO.LOW)

display\_message("Access Granted", f"ID: {id}", delay=3)

print(f"Access Granted for ID: {id}")

time.sleep(10) # Keep solenoid open for 5 seconds

else:

GPIO.output(SOLENOID\_LOCK\_PIN, GPIO.HIGH) # Keep the lock closed

GPIO.output(ACCESS\_GRANTED\_PIN, GPIO.LOW)

GPIO.output(ACCESS\_DENIED\_PIN, GPIO.HIGH)

display\_message("Access Denied", f"ID: {id}", delay=3)

print(f"Access Denied for ID: {id}")

time.sleep(10) # Keep access denied signal for 5 seconds

# Reset GPIO states

GPIO.output(SOLENOID\_LOCK\_PIN, GPIO.HIGH) # Ensure the lock is closed

GPIO.output(ACCESS\_GRANTED\_PIN, GPIO.LOW)

GPIO.output(ACCESS\_DENIED\_PIN, GPIO.LOW)

time.sleep(2) # Adjust sleep time as needed

except KeyboardInterrupt:

pass

# Function to monitor fire sensor

def fire\_sensor\_monitor():

try:

while True:

# Read the state of the sensor

sensor\_state = GPIO.input(FIRE\_SENSOR\_PIN)

if sensor\_state == GPIO.LOW:

print("Fire detected!")

display\_message("Fire detected!", "", delay=3)

# Activate all outputs

GPIO.output(POWER\_SUPPLY\_PIN, GPIO.LOW) # Inverted logic: LOW means on

GPIO.output(EMERGENCY\_LIGHT\_PIN, GPIO.HIGH)

GPIO.output(WATER\_PUMP\_PIN, GPIO.LOW) # Inverted logic: LOW means on

# Print and display message when water pump is on

print("Water pump on")

LCD1602.write(0, 1, "Water pump on")

else:

# Ensure all outputs are turned off

GPIO.output(POWER\_SUPPLY\_PIN, GPIO.HIGH) # Inverted logic: HIGH means off

GPIO.output(EMERGENCY\_LIGHT\_PIN, GPIO.LOW)

GPIO.output(WATER\_PUMP\_PIN, GPIO.HIGH) # Inverted logic: HIGH means off

# Wait for 1 second before checking again

time.sleep(1)

except KeyboardInterrupt:

pass

def main\_lcd\_monitor():

try:

while True:

gas\_detected = not GPIO.input(GAS\_SENSOR\_PIN)

fire\_detected = GPIO.input(FIRE\_SENSOR\_PIN) == GPIO.LOW

if not gas\_detected and not fire\_detected:

LCD1602.write(0, 0, "EVERYTHING OK")

LCD1602.write(1, 0, "")

time.sleep(3)

else:

time.sleep(1)

except KeyboardInterrupt:

pass

# Create threads for gas sensor, RFID, fire sensor, and main LCD monitor

gas\_thread = threading.Thread(target=gas\_sensor\_monitor)

rfid\_thread = threading.Thread(target=rfid\_sensor\_monitor, args=(reader,))

fire\_thread = threading.Thread(target=fire\_sensor\_monitor)

lcd\_thread = threading.Thread(target=main\_lcd\_monitor)

# Start all threads

gas\_thread.start()

rfid\_thread.start()

fire\_thread.start()

lcd\_thread.start()

# Display initial message

display\_message("INDUSTRIAL", "SAFETY", delay=3)

# Wait for threads to finish (should never happen in this case)

gas\_thread.join()

rfid\_thread.join()

fire\_thread.join()

lcd\_thread.join()

# Cleanup GPIO

GPIO.cleanup()