**tutorial 3**

**Main task (code)**

Design a program for an embedded system that meets the following requirements.

1. A report of the states of gas detector and over temperature detector be sent to the PC when keys 2 and 3 are pressed respectively.

#include "mbed.h"

// Analog inputs from sensors

AnalogIn airQuality(A0); // Gas detection

AnalogIn tempProbe(A1); // Temperature reading

// Key inputs

DigitalIn gasButton(D3, PullDown); // Button to check gas

DigitalIn tempButton(D2, PullDown); // Button to check temperature

// Serial interface to PC

UnbufferedSerial terminal(USBTX, USBRX, 115200);

// Output message buffer

char outputLine[128];

// Serial print helper

void printMessage(const char\* text) {

terminal.write(text, strlen(text));

}

// Get gas sensor value in %

int gasPercent() {

return (int)(airQuality.read() \* 100.0f);

}

// Get temperature in Celsius

int tempCelsius() {

return (int)(tempProbe.read() \* 3.3f \* 100.0f);

}

int main() {

printMessage("System is ready Press ((Key1) for Temperature or ((Key2) to chech Gas.\r\n");

while (true) {

if (gasButton.read()) {

int gas = gasPercent();

snprintf(outputLine, sizeof(outputLine), "Gas Level: %d%%\r\n", gas);

printMessage(outputLine);

while (gasButton.read());

ThisThread::sleep\_for(200ms);

}

if (tempButton.read()) {

int temp = tempCelsius();

snprintf(outputLine, sizeof(outputLine), Temperature: %d°C\r\n", temp);

printMessage(outputLine);

while (tempButton.read());

ThisThread::sleep\_for(200ms);

}

ThisThread::sleep\_for(100ms);

}

}

1. Send the current alarm state to the PC at regular intervals

#include "mbed.h"

// Sensor channels

AnalogIn gasInput(A0);

AnalogIn tempInput(A1);

// Serial communication to PC

UnbufferedSerial terminal(USBTX, USBRX, 115200);

// Background system

EventQueue scheduler;

Thread worker;

Ticker periodicAlarm;

// Output buffer

char alarmBuffer[128];

// Serial output wrapper

void logToPC(const char\* text) {

terminal.write(text, strlen(text));

}

// Convert sensor values

int readGas() {

return static\_cast<int>(gasInput.read() \* 100.0f);

}

int readTemp() {

return static\_cast<int>(tempInput.read() \* 3.3f \* 100.0f);

}

// Evaluate alarm condition

const char\* checkAlarmStatus(int temp, int gas) {

return (temp > 40 || gas > 60) ? "WARNING" : "SAFE";

}

// Alarm reporting task

void pushAlarmInfo() {

int gasVal = readGas();

int tempVal = readTemp();

const char\* state = checkAlarmStatus(tempVal, gasVal);

snprintf(alarmBuffer, sizeof(alarmBuffer),

"[ALERT] Temp: %d°C @@@ Gas: %d%% → Status: %s\r\n",

tempVal, gasVal, state);

logToPC(alarmBuffer);

}

int main() {

worker.start(callback(&scheduler, &EventQueue::dispatch\_forever));

logToPC("System active. Alarm updates every 3 seconds.\r\n");

periodicAlarm.attach([] {

scheduler.call(pushAlarmInfo);

}, 3s);

while (true) {

ThisThread::sleep\_for(500ms);

}

}

1. Continuously send data to the PC regarding the alarm state, temperature readings, and gas detection status.

#include "mbed.h"

// Analog sensor inputs

AnalogIn gasPort(A0);

AnalogIn tempPort(A1);

// Serial link

UnbufferedSerial pcSerial(USBTX, USBRX, 115200);

// System backend

EventQueue taskQueue;

Thread serviceThread;

Ticker dataTicker;

// Output container

char systemReport[256];

// Serial sender

void sendLog(const char\* line) {

pcSerial.write(line, strlen(line));

}

// Readings

int getGas() {

return static\_cast<int>(gasPort.read() \* 100.0f);

}

int getTemp() {

return static\_cast<int>(tempPort.read() \* 3.3f \* 100.0f);

}

// Determine status

const char\* systemState(int temp, int gas) {

return (temp > 40 || gas > 60) ? "WARNING" : "SAFE";

}

// Composed report

void generateReport() {

int gas = getGas();

int temp = getTemp();

const char\* state = systemState(temp, gas);

snprintf(systemReport, sizeof(systemReport),

"\r\n@@@ Live System Data @@@\r\n"

"Temperature: %d°C\nGas: %d%%\nStatus: %s\r\n",

temp, gas, state);

sendLog(systemReport);

}

int main() {

serviceThread.start(callback(&taskQueue, &EventQueue::dispatch\_forever));

sendLog("System active. Alarm updates every 3 seconds...\r\n");

dataTicker.attach([] {

taskQueue.call(generateReport);

}, 5s);

while (true) {

ThisThread::sleep\_for(1s);

}

}

1. Trigger a warning on the PC interface if temperature or gas levels exceed safe limits.

#include "mbed.h"

// Analog sensor input

AnalogIn gasRead(A0);

AnalogIn tempRead(A1);

// Serial communication

UnbufferedSerial debugPort(USBTX, USBRX, 115200);

// Message formatting

char lineBuffer[256];

// Output wrapper

void displayToPC(const char\* msg) {

debugPort.write(msg, strlen(msg));

}

// Sensor readings

int gasPercent() {

return static\_cast<int>(gasRead.read() \* 100.0f);

}

int tempCelsius() {

return static\_cast<int>(tempRead.read() \* 3.3f \* 100.0f);

}

int main() {

displayToPC("Realtime environment check initialized...\r\n");

while (true) {

int gas = gasPercent();

int temp = tempCelsius();

// Display readings

snprintf(lineBuffer, sizeof(lineBuffer),

"\r\nTemperature:: %d%% [[[[]]]] Gas: %d°C\r\n", temp, gas);

displayToPC(lineBuffer);

// Warning condition

if (gas > 60 || temp > 40) {

displayToPC("WARNING: Unsafe temperature or Gas detected!!!!!!!\r\n");

}

ThisThread::sleep\_for(1s);

}

}