COMP2004 C1 W2

Semester 2 2022

# Key Information

You MUST complete the following forms so your work can be assessed.

|  |  |
| --- | --- |
| TASK |  |
| GitHub Name | tcollyer1 |
| IoT Central URL | <https://comp2004-cw2-tcollyer.azureiotcentral.com> |
| I confirm I have given the tutor access to my IoT central for the purpose of assessment |  |
| Demo Video URL |  |
| Submission TAG |  |

Please state which requirements you have attempted. Indicate how this can be evidenced

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Requirement | Not done | Partially Completed | Completed | Comments (optional) |
| 1 |  |  |  | Sensor data & time collected every 10s in **main.cpp**, sampling jitter minimised by taking multiple samples and averaging. On its own high priority thread. Sensor data class found in **SensorData.h** |
| 2 |  |  |  | Data is written to SD card every minute in **main.cpp**, values are read from buffer and written using Buffer and SDWrite classes (**SDWrite.h**, **Buffer.h**) |
| 3 |  |  |  | FIFO buffer is used (**Buffer.h**) to store sensor data and its contents are written to the SD card every minute. Writing to the FIFO is non-blocking, and uses virtual functions for Mbed-specific features (such as the button, LED and ticker). Red LED is lit if buffer is full. |
| 4 |  |  |  | Upper and lower limits for temperature in **main.cpp**. Collected sensor data is checked against these and alarm is written to terminal if they move outside the limits. Pressing blue user button prompts the alarm to stop firing for a minute using signal-wait – on thread 3. |
| 5 |  |  |  | 4 threads used – for writing to SD and reading buffer, collecting sensor data, communicating with and sending data to Azure and waiting on user’s press of blue button. |
| 6 |  |  |  | Signal-wait mechanisms and timer interrupts used to minimise power consumption – timer interrupts on all 3 threads, signal-wait on blue button press handling (**PushSwitch.h**), signal-wait when sending to Azure. |
| 7 |  |  |  | Remote functions implemented in **main.cpp**, with commands implemented on Azure to call these functions. Appropriate responses are returned. |
| 8 |  |  |  | Sensor measurements and date/time data are sent to Azure every 10 seconds using the newest item added to the buffer. |
| 9 |  |  |  | Mutex lock used within the buffer, however semaphores were not fully implemented. |
| 10 |  |  |  | Critical errors cause the red LED to light and an error to be generated to the terminal. |
| 11 |  |  |  | Files are structured and indented with comments. Classes are in their own files and name can be found at the top of each source file. |

(Over…)

# Verification Record (results)

You are responsible to evidence each requirement as robustly as possible. Please provide evidence of each requirement in the space below.

Some options include:

* A set of instructions for the tutor to follow
* Small modifications to the source code
* Logging data to the terminal
* Some might require code inspection

Try to be concise (I do NOT want an essay!).

Some requirements may be more challenging to verify than others. You may add pre-compiler directives to assist with this.

If there are problems *fully* verifying a requirement (and there will be!), justify why in the spaces below.

You do not need to comment on requirement 11

## Requirement 1

This device shall periodically measure sensor data at a **fixed** and **deterministic** rate. This shall include temperature (deg C), pressure (mbar) and light levels (from the LDR). The default update rate shall be once every 10 seconds and you should write your code to **minimize sampling jitter**. The data shall be encapsulated in a single C++ structure or class. It is suggested that sampling is performed on the highest priority thread.

|  |
| --- |
|  |

## Requirement 2

With the SD card inserted and mounted, the date, time and sensor data shall be written to text file on the SD card *in a format that is human readable and easy to edit (with a text editor)*. See requirement 3 for timing of this. Time and date should be acquired via a network time server.

|  |
| --- |
|  |

## Requirement 3

SD card writes are slow, consume significant power and cause wear to the flash memory they contain, so internal memory buffering should be used to reduce the number of SD card write cycles. Data should be (FIFO) **buffered** in internal memory and only written to the SD card in blocks (once an hour being the slowest, and once a minute the most frequent).   
To facilitate this, you should write, use and test a **thread-safe** FIFO buffer. This shall be written as a C++ class that encapsulates the functionality needed to buffer data records between writes. It shall also encapsulate all the necessary thread synchronization.   
  
- Writing to the FIFO should be non-blocking (return an error code if it fails)  
- Reading from the FIFO should be blocking if the buffer is empty  
- You are to use the producer-consumer pattern as described in the lecture slides  
- For full marks, maximise reusability by using templates and avoiding tight-coupling to platform specific objects (consider using closures and/or pure virtual functions for the platform-specific behaviours).  
  
It should be possible to detect if the buffer becomes full, and in such cases, you should log an error and light a red LED. An empty buffer is not an error.  
  
Do NOT use code from other sources except the provided template and my lab code. All submitted code will be checked using plagiarism detection tools. See lecture slides on the producer-consumer pattern for more guidance.

|  |
| --- |
|  |

## Requirement 4

There shall be upper and lower thresholds for temperature (), pressure and light . An alarm message should ~~sound~~ be written to the serial terminal if any measurement moves outside these threshold values. The blue user button can be pressed to cancel the alarm for 1 minute. **Note** - The buzzer cannot be used due to a technical fault, so you should **mock it**.

|  |
| --- |
|  |

## Requirement 5

No operation in this project should interfere with (i.e. block) any other. Therefore, your code **must** be multithreaded and use appropriate synchronization to prevent data inconsistency or corruption. You should use *at least* 3 separate and dedicated threads for (i) writing to the SD card; (ii) communicating with the serial interface (iii) communicating with the network. Event Queues are recommended but not a requirement. In addition, you may also use interrupts if appropriate and where justified, again with suitable synchronization.

|  |
| --- |
|  |

## Requirement 6

You should write your code to minimize power consumption where possible. Where busy-wait loops are used, you are required to justify why in the code comments.

|  |
| --- |
|  |

## Requirement 7

For full marks, you should use the Azure portal communicate with the target board to perform certain critical actions via remote function calls (see Lab Task 396). See Table 2 lists the functions you should implement. If you are unable to do this, then you may use a serial terminal (limited max 50% of this mark)

|  |
| --- |
|  |

## Requirement 8

Your device should send each date, time and set of sensor measurements to the Azure IoT center (see Task 396 and lab notes). For partial marks, you can host the information on an internal web page (accessible from the PC) as demonstrated in lab Task-394.

|  |
| --- |
|  |

## Requirement 9

You shall mitigate against the occurrence of deadlocks and/or thread starvation using timeouts with any blocking functions (see the documentation on classes such as Semaphore or Lock). Any timeouts that occur should be considered a “critical error”

|  |
| --- |
|  |

## Requirement 10

Should a critical error occur, the red LED should light, and an alarm should sound for 30 seconds. The system should then reset itself. Details of any errors should ALWAYS be logged to the serial interface (if at all possible). You will need to find ways to induce critical errors so that this can be tested and demonstrated.

|  |
| --- |
|  |