FIT2070 Assignment 2 Report

Callum White
24571520

This report is to be read in addition to running the included C program. In this report I will explain how the program works and some of the thread-related issues that I overcame during the development of the program.

My first version of the assignment simply had two threads which were constantly iterated over. One thread generated the light, temperature and humidity values and saved them to global variables. The second thread took them out of the global variables and printed them. There was a single semaphore which would control access to the resources. I soon found that this was not a very good solution, as it was very easy to either miss printing values that were generated by the first thread, or print the same values multiple times. This issue became more obvious when I put sleeps in either of the threads (outside of the resource semaphores).

From here, I decided to change the second thread to make it save all values to a set of circle buffers. A third thread would then take the values out of the oldest unprinted position of each buffer, compute the aggregate and average, and print them out. This way, the circle buffers would help with the issue of when multiple values are generated before any values can be printed, as they would simply keep being added to the buffer. The only time we then have a problem is when the buffer eventually gets full and starts overwriting valid values that have not yet been printed. This issue can be overcome by intelligently choosing how large the circle buffers should be, based on how long each thread takes to run, and how long it takes for this issue to occur.
Unfortunately, with this implementation I was unable to get the operation of the

threads synced correctly when using semaphores. I was experiencing very inconsistent program output behaviour. I would get floating point exceptions 50% of the time I ran the program, and on other times 0's would be printed out for some values when they were not supposed to be 0.

After a lot of unsuccessful debugging of these issues, I decided to implement it using busy-wait's instead. By doing this I felt I had more control over the conditions of when certain code is run. Because of this, I think the circle buffers are not utilised as well as they could be and the program is not as efficient as it would be if implemented with semaphores. However, I was able to get a working program out of this implementation.

The submitted code has a 2 second sleep in the third thread which slows down the output of the code enough for us to have time to follow each individual output. When run without this 2 second sleep, the program still runs as per usual. It just runs way too fast for us to inspect the output.