NWEN303 Concurrent Programming (2016)
Assignment 1 : Basic Concurrency
(Due midnight, Wednesday 27 July)

This assignment is intended to help you to understand some basic aspects of concurrency, and to give you experience in implementing multi-threaded programs in Java and measuring their performance.

You may work in pairs for the programming and data collection parts of questions 2 and 3, so long as you indicate who you worked with. Your answer to question 1, and the summaries you write for questions 2 and 3, must be your own work.

You should submit your work using the online submission system, giving your answers as a pdf document (scan it if necessary), and including the code you used for questions 2 and 3.

Note that the marks shown are provided as a guideline, and may be changed.

1. **Synchronisation** [20 marks]

   Suppose you wish to find some natural number, k, such that $f(k) = 0$, for a given function f, assuming that at least one such k exists.

   The following is a rather poor attempt at constructing a concurrent algorithm to solve this problem using N processes, where the intention is that process i checks $f(i), f(i+N), f(i+2N), \cdots$, and *one* process which finds a k such that $f(k) = 0$ prints that value of k:

Algorithm: Concurrent search (N processes)
integer k boolean $found := false$
p(i), for $i = 0, \cdots, N-1$

$k := i$
while not $found$
  if $f(k) = 0$ then
  $found := true$
  print k
  else $k := k + N$

   (a) [6 marks] Explain, with reference to the above algorithm, what is meant by a *race condition*. Identify the race conditions that exist in this algorithm and describe the incorrect behaviours that they can cause.

   (b) [6 marks] Describe how you would modify the above algorithm to avoid race conditions, explaining carefully how you would ensure that the algorithm behaves correctly without suffering from either race conditions or deadlock.

   (c) [4 marks] Explain what is meant by *fairness*, and how the correctness of your algorithm from part **(b)** depends on a fairness assumption.

   (d) [4 marks] Briefly describe one way in which your algorithm from part **(b)** could be modified so that it would work correctly in the absence of fairness. Discuss the impact that this approach has on the performance of the algorithm.

2. **Measuring Array Sum** [40 marks]

   On the NWEN303 assignments web page, there is a version of the Array Sum program discussed in lectures, instrumented to print timing data. The program takes the number of threads (`N`) and number of array elements (`M`) as command line arguments. It then sums the array without using threads, and then using `N` threads, and prints the time take for each.

   You are to run the program with varying array sizes and numbers of threads, and on machines with different numbers of processes/cores (including Lighthouse, which can run 64 threads). You should also change whether array elements are added directly to the global `sum` variable or using a thread local variable, to see the effect of contention on the shared variable. Finally, you should try to work out how much time is spent creating threads and how much doing the actual summation.

   Write a summary of your findings, showing timings you recorded and explaining how the various factors you varied affect the time taken to sum the array.

3. **Recursive Array Sum** [40 marks]

   An alternative approach to summing an array using multiple threads is to successively divide the array into halves, giving each half to a new thread to sum, until the segment a thread is asked to sum is small enough to sum directly. Thus, the `run` method tests the size of the array segment described by `lo` and `hi` — if it is below some threshold, if sums the segment using a loop; otherwise it creates two new threads, passes half of the array to each thread, then waits till they have both finished and adds together the sums they computed.

   You are to implement an array sum program based on this approach, and experiment with the size of the array, the thresh-hold below which the array segment is summed directly, and the number of processes/cores available, etc., to see how they affect the performance of the program. Write a summary of your findings, comparing them with the results you obtained in question 2.

   As an optional extra, you may implement this version using suitable classes from the Java concurrency library, but you should implement a version using explicit threads first.

   Total possible: 100 marks