ECE421 - Assignment 3

A Multi-threaded Sort

Concurrency can be applied to many systems to speed up the application. Concurrency can also be utilized to produce programs with a clearer structure, identifying common components, which can act as independent threads of control within the entire process space.

Sorting is a computationally expensive activity, and we will often seek to abandon a simple sequential solution when we have multiple processors at our disposal.

For this initial attempt, lets assume that we have an infinite number of processors available, each with an infinite resource base, in terms of memory, file descriptors, etc., i.e. our ideal solution is the solution, which **maximizes the concurrency** within the solution.

Your code should accept **a duration** and **a, potentially large, number of objects**. The program should then sort the **objects** using a **multi-threaded sort algorithm**, which takes advantage of our infinite set of computational resources. Despite the 'highly tuned' solution, the **duration** specifies the maximum amount of time the algorithm has to complete; if this **duration** is exceeded, the program must stop, tidy-up and return control to the calling method.

You might find the attached document useful when considering your solution to this problem, i.e. unless you can find a better solution – use it! Please Note: the document provides an algorithm – one line of algorithm DOES NOT necessarily equal one line of code!

To assist you with this task, please ponder (and answer) the following questions as you go …..

1) What is your definition of an object?
2) What strategies should be deployed in terms of accepting input (i.e. the large number of objects.)?
3) Is your sort generic? What sorting criterion does your system use? Is this criterion flexible; i.e. changeable by the user at the point of execution? What limitations have you placed upon legal sorting criteria? To make it reusable to other people, how should we include it into the Ruby Class hierarchy?
4) In reality we have a uniprocessor system, describe "equationally" what is happening to your solution as you increase the concurrency (e.g. produce a regression model (remember regression from Statistics) of processing time against the number of threads. Your solution can be modeled as a program which: (1) has a component which produces threads; and (2) a set of threads which undertake the same (small) task. This is in essence the basis of stress testing (discussed in ECE 322)

5) Concurrent systems tend to crash frequently – what approach to exception-handling have you devised? Consider the content of the library at: http://c2.com/cgi/wiki?ExceptionPatterns; which are applicable to this problem? Is Module Errno useful in this problem? What components of the Ruby exception hierarchy are applicable to this problem? Discuss in detail your strategy for exception-handling.

6) What differences exist between thread-based and process-based solutions? How has this impacted the design of your solution?

7) Do you have any race-condition or task synchronization concerns about your solution? How do we tidy-up a multi-threaded program, if stopped mid-execution?

8) As discussed in CMPUT 301: What is configuration management? What is version control? Are you using either concept? If "yes", describe your process and any tool support what you utilize – illustrate your process with regard to Assignments 1 and 2; if "no", justify your decision.

9) Briefly Explain:

   a. What is refactoring (as discussed in ECE 325 / CMPUT 301)?
   b. Are you using refactoring in your development process? Justify your answer?
   c. If "yes", give examples, minimum of 2, of the refactoring "patterns" that you used in Assignment 1
   d. If "no", give examples of where your solution to Assignment 1 would be improved by applying refactoring patterns. Supply a minimum of two different (i.e. different refactoring patterns) as examples.

This assignment is worth 20% of the total marks of the ECE421 assignments; and must be completed in three parts

Part 1: Your rationale for your design decision must be completed by" **Tuesday 1st March @ midnight**

Part 2: Your System Design (written as a set of assertions) must be completed by: **Tuesday 1st March @ midnight**

Part 3: Your system must be completed by: **Tuesday 1st March @ midnight** and must be ready for demonstration in the practical session.

In addition to the practical demonstration, you are required to hand-in:

1) A detailed rational for your augmented decisions with regard to the above design questions.
2) A list of deviations in the contracts implemented form the contracts specified in Part 1.
3) A copy of the code
4) A description of any additional testing beyond that described by your contracts.
5) A list of known errors, faults, defects, missing functionality, etc ….telling us about your system's limitations will score better than letting us find them!

Please hand in both components and hence all sub-components by definition must be machine readable. In addition,

- The Subject Line should be in a specific format - Assignment 3: Group 1 [, Part 1...]

- Filetypes should be only of .doc or .pdf

- Filenames for code should be in ruby format (all lower cases, with words concatenated with a dash)

- For each assignment with a code section, there must be a primary executable file that is the main entry for other codes. This has to have the format: assignment3_main.rb (where 3 corresponds to the appropriate number in the assignment sequence). This main file should also contain a comment section with a list of the group members if necessary and some description of the runtime requirement of the code (e.g. commandline activation format)