Carleton University
Department of Systems and Computer Engineering
SYSC 3303 - Real-Time Concurrent Systems – Winter 2014
Assignment 2

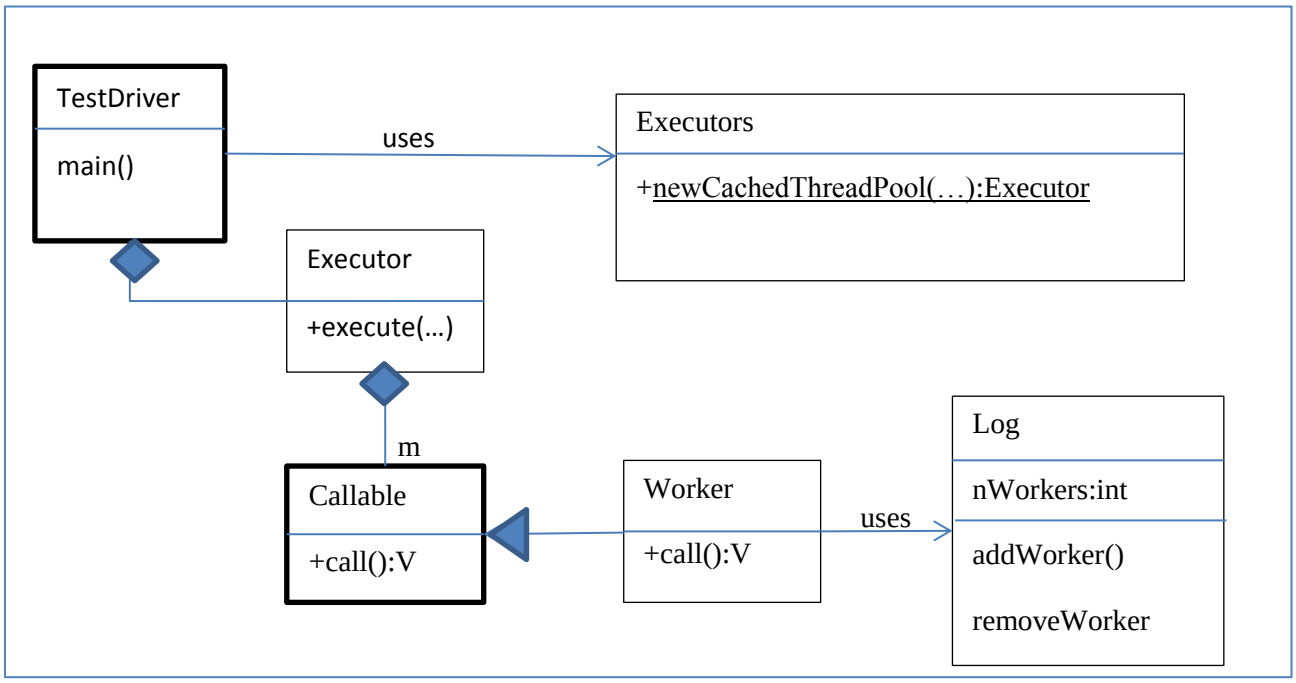
Assignments are to be completed individually, not in your project teams. This document is descriptive not prescriptive – ask questions in class or in a discussion forum if you need more information.

Objectives:

- Use of Java thread interactions and util.concurrent.Executor
- Use of Xenomai mutex and condition variables
- Testing the behavior of concurrent systems

Part A Java Thread Interactions

You will write a program for the system described in the class diagram below. It is a combination of the Executor lesson(s), the Future lesson, and the Synchronize lesson in the **Concurrent Animated** project.



The TestDriver creates a CachedThreadPool executor and then initiates a series of test-cases for that executor. Like **ConcurrencyAnimated**, a test-case consists simply of a number *n*, where that number *n* is the number of variable-length "jobs" to be executed. (See the three Executor lessons in the **ConcurrentAnimated** project). The job passed from the executor to the worker will include a unique job identifier and an expected (variable/random) execution time. Unlike the Executor lessons, your executor will use Callable workers (See the Future lesson) where each worker will return the actual time

taken to complete their assigned job identifier. While doing their "work", each worker will access a shared resource called Log. The log object will contain a count of the number of workers currently working. (See the Synchronized lesson). The executor should periodically (depending on the resolution of your spin cycles) check and report statistics about itself (Study the API for ThreadPoolExecutor).

Upon completion, your system should provide a statistical summary of the program's "behavior" during each test case:

- For each job dispatched, the number of jobs dispatched, their expected duration and their actual duration
- The number of active threads before and after the test case.

Requirements:

1. You must write the project in Java.
2. The Log must be a passive thread-safe object (its methods must be synchronized).
3. Omitted from the diagram is the object responsible for gathering the Future return-values from the workers. You may assign this responsibility to an existing class, or create your own class. If you have trouble with Callable and Future, simply ignore this requirement and use Runnable objects. You will be missing the data for the actual duration of that job but at least you will have a working system.
4. Your program must run long enough and have a test case showing that idle workers are decommissioned by the executor.

Suggestion: Build the system incrementally. Work either left-to-right (top-down) or right-to-left (bottom-up). Keep the TestDriver class very simple at first – use it to simply create all the objects needed at startup; later, add code as a means to automate different test cases.

Part B Xenomai

The MAE CUSP project is building an aircraft simulator with nine-degrees of freedom, meaning that the pilot will sit inside a giant ball and will be able to simulate fully rolling a plane. The orientation of the ball is measured using sensor fusion: two sensors with two different capabilities are used and their sensed values are "fused" mathematically (e.g averaged) to generate an accurate *fused orientation value*. Without going into a lot of technical detail[1] !:

1. The accelerometer actually measures inertial acceleration from which can be computed the orientation. For our purposes, it is important to note that its measurements are relatively fine but suffers from drift over time and thus requires correction.
2. A gyroscope provides very quick samples, but the samples are inputs to a time-series computation - the average over the last *n* samples.

Note: No math required in this program. The math is only explained a bit to get you interested and to make the assignment relevant.

Motors must be updated at 60 Hz. The *fused orientation value* must be updated at this frequency as it is an input to the motor-drive logic.

Your program will have threads:

1. Accelerometer thread: To generate a "sample" at the rate of 30 Hz
2. The gyroscope thread: To generate "samples" at the rate of 300 Hz.
3. The sensor-fusion thread: To compute the orientation at the rate of 60 Hz, accessing the samples in a thread-safe manner.

Your program will:

1. Be written in Xenomai/C
2. Use rt_mutex as the only synchronization mechanism.
3. Provide output showing all accesses were thread-safe. You could potentially show that no accesses were not thread-safe !?.
4. No printing must be done during the program execution. All printing must be done at the end of the program's run. In other words, log, don't print.

[1] Join the CUSP project for your 4th year project!

Common Instructions

1. Each program must be accompanied by a README.txt that describes how to run your program
2. Each program should be accompanied by a "final report" (partA.pdf and partB.pdf).
3. Each "final report" should contain
    a. A description of your testing strategy. How are you generating your inputs? How are you instrumenting your code? Any deviations from the assignment's design
    b. A summary of your results: A data printout – or an excerpt – or a calculation (e.g. average, min, max) of the data, even the presence or absence of data. Whatever representation, the data should support a conclusion that your code is threaded and thread-safe.

Assignments are to be submitted electronically on CULearn Course page. Emailed submissions will not be accepted. See the course outline for the procedure to follow if illness causes you to miss the deadline.

Marking Scheme coming …