University of Glasgow | School of Computing Science

Assessed Coursework

Course Name	Advanced Operating Systems (M)			
Coursework Number	Exercise 3			
Deadline	Time:	11:00am	Date:	13 March 2014
% Contribution to final course mark	12%			
Solo or Group ✓	Solo	✓	Group	
Anticipated Hours	12			
Submission Instructions	Submit via drop-box outside Teaching Office			
Please Note: This Coursework cannot be Re-Done				

Code of Assessment Rules for Coursework Submission

Deadlines for the submission of coursework which is to be formally assessed will be published in course documentation, and work which is submitted later than the deadline will be subject to penalty as set out below.

The primary grade and secondary band awarded for coursework which is submitted after the published deadline will be calculated as follows:

(i) in respect of work submitted not more than five working days after the deadline
   a. the work will be assessed in the usual way;
   b. the primary grade and secondary band so determined will then be reduced by two secondary bands for each working day (or part of a working day) the work was submitted late.

(ii) work submitted more than five working days after the deadline will be awarded Grade H.

Penalties for late submission of coursework will not be imposed if good cause is established for the late submission. You should submit documents supporting good cause via MyCampus.

Penalty for non-adherence to Submission Instructions is 2 bands

You must complete an "Own Work" form via https://webapps.dcs.gla.ac.uk/ETHICS for all coursework UNLESS submitted via Moodle

Advanced Operating Systems (M): Exercise 3

Dr Colin Perkins

25 February 2014

In the last few lectures, we have considered message passing libraries and programming languages as a conceptually appropriate method for programming multicore systems. The aim of this assessed exercise is to investigate programming using such message passing systems. You should complete all parts of this exercise, which is worth 12% of the marks for this course.

Background: The prototypical message passing system is the Erlang programming language, a functional language designed around the concept of message passing actors that runs on it's own virtual machine. An alternative to Erlang is the Akka message passing library for the Java virtual machine. This can be used with either Java or the Scala programming language, and provides an actor-based message passing library built using Java threads. Both Erlang and Akka are open source software, and run on readily available operating systems such as Linux, Mac OS X, and Windows.

The latest stable release of the Erlang programming languages is Erlang/OTP R16B03. This is available to download from `http://erlang.org/` (Windows binaries are available on that site; alternatively the compiler and runtime cleanly installs into a home directory on the Linux systems in the Level 4 lab if compiled with `./configure --prefix=$HOME && make`). Akka v2.2.3 is available as a Java archive (.jar) file that can be downloaded from `http://akka.io/` (Akka v2.3 was in the final stages of testing as this exercise was being prepared; it does not look to make any significant changes that would impact this exercise, and can be used if available). Akka has bindings for both Java and for the Scala programming language (`http://scala-lang.org/`). Both Erlang and Akka have extensive documentation available from their respective web sites, and there is also an Erlang tutorial available at `http://learnyousomeerlang.com`.

A key question with these message passing systems is how easy do programmers find it to adapt to using the new language and/or message passing library in place of threads and lock-based concurrency? If working programmers cannot easily transition to using message passing libraries and languages, the adoption of these systems will be slow, and the claimed benefits of message passing systems might not be realised. In this exercise you will test this by writing a simple program using one of these message passing systems, and reflect on your experience.

Question: Using your choice of either the Java programming language with the Akka message passing library, the Scala programming language with the Akka message passing library, or the Erlang programming language, write a message passing program that can perform temperature conversions as described below. You will need to spend some time reading up on these platforms to decide which best fits with your experience, and to learn how to use the features that it provides, before you decide which to use, and start the exercise.

Your program should consist of a temperature converter actor, a display actor, two sensor actors, and a clock actor. The temperature converter actor should respond to two types of message:

- If sent a `ConvertToCelsius` message with a temperature in degrees Fahrenheit as an argument, it should reply with a `Temperature` message containing the temperature in Fahrenheit and the equivalent temperature in Celsius.

```
                       +--------+
                 +-----| Sensor |<----+
                 |     +---+----+     |      +-------------+
+---------+      |         |          +----->|             |
|         |<-----+     +---+---+             | Temperature |
| Display |            | Clock |             |  Converter  |
|         |<-----+     +---+---+      +----->|             |
+---------+      |         |          |      +-------------+
                 |     +---+----+     |
                 +-----| Sensor |<----+
                       +--------+
```

Figure 1: Message flow between actors

- If sent a `ConvertToFahrenheit` message with a temperature in degrees Celsius as an argument, it should reply with a `Temperature` message containing the temperature in Celsius and the equivalent temperature in Fahrenheit.

A temperature in Fahrenheit can be converted to Celsius by subtracting 32 then multiplying by 5/9. A temperature in Celsius can be converted to Fahrenheit by multiplying by 9/5 then adding 32.

The two sensor actors should send periodic `ConvertToCelsius` and `ConvertToFahrenheit` messages to the temperature converter actor, each time they receive a `Tick` message from the clock actor, as if they were repeatedly measuring the temperature of some system (the actual values of the temperature that are converted are unimportant for this exercise). The clock actor should send one tick per second to each of the two sensors. When a sensor receives a reply with the converted temperature, it should send it to the display actor. The display actor should wait for `Temperature` messages, and print out the temperate in both Fahrenheit and Celsius (your program does not need a fancy user interface, simple textual output is sufficient). The five actors therefore pass messages between themselves as shown in Figure 1. The actors may need to be driven by some form of main program, depending on the message passing system used; you should write whatever supporting infrastructure is needed.

Once you have written and tested your actor system, you should prepare a short report (not more than two sides of A4 paper) reflecting on the process of building your message passing temperature converter system. Your report should focus on:

- Your choice of programming languages and message passing library. Why did you make your choice, and what did you believe to be the advantages and disadvantages of this system at the start of the exercise.

- The installation process, documentation, and learning curve for your chosen languages and message passing library. Discuss the main challenges in installing and learning your chosen system.

- The design of your temperature converter system. Describe any unusual features of your design, and any implementation challenges you faced.

- Your overall opinion of the system. Do you think your chosen message passing system is mature enough for non-expert programmers to use?

The aim of the report is to record your experiences and reflections on using your chosen system. It will be marked on the quality of written argument, and for critical reflection on the issues noted above. There is no single correct answer for this exercise, and no expectation that the systems will work effectively, or that particular problems will be experienced. Give, and justify, your opinion of the system, reflecting on the issues and challenges you faced. One third of the marks for the exercise are available for your code; two thirds for your report.

Submission: A printed copy of your report must be submitted, along with a print out of your source code, by 11:00am on 13 March 2014. As per the Code of Assessment policy regarding late submissions, submissions will be accepted for up to 5 working days beyond this due date. Any late submissions will be marked as if submitted on time, yielding a band value between 0 and 22; for each working day the submission is late, the band value will be reduced by 2. Submissions received more than 5 working days after the due date will receive an H (band value of 0).

A drop box will be available for submissions in outside the Teaching Office in Lilybank Gardens, and submissions will only be accepted via that drop box. This problem set is worth 12% of the mark for this course. Ensure that your name and matriculation number are included on each submission, and that you have submitted a statement of originality. Submissions that do not follow these submission instructions will be penalised two bands.