Individual Task

Concurrent Systems SE3CS12 - Spring 2016

Coursework I : Modelling Concurrent Systems

Objective

The overall objective of this coursework is to develop an appreciation of the basic concepts of modelling concurrent systems through an implementation of a simplified version of a real-world application. An additional objective is to develop practical knowledge of concurrent system design and implementation in concurrent programming languages like JCSP/Jibu.

The coursework consists of two parts:

1. Implementing a Concurrent system (20% of coursework mark)
    - The purpose of this task is to develop modelling concepts in the CSP formalism by exploring the design and implementation of a concurrent application.

2. Investigate concurrency concepts in Modern OO languages (5% of coursework mark)
    - The purpose of this task is develop further insight into emerging concurrency concepts such as "lock-free" data structures.

Marking : This is a major piece of work which constitutes 25% (out of 30%) of the coursework marks component for the module.

Submit the write-up of the individual task, including both parts as one full writeup, through the **assignment folder on Blackboard**.

The deadline is **17th March 2016**.

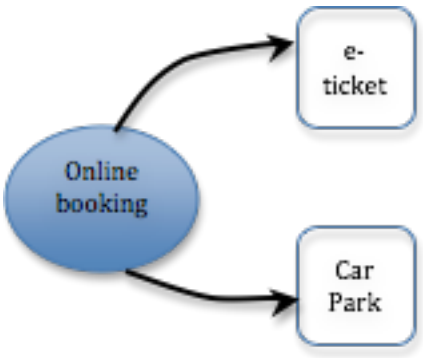
1. Implementing a Concurrent System

Online Car Park booking system scenario:

Online booking systems for airport parking is a modern day phenomena arising from the Internet and WWW application software! The aim of this implementation is to gain insights into a CSP model of an online booking system. The system can be modelled as three components:

- An online reservation system (web interface)
- An email system for e-ticket
- The car park itself

At the highest level the complete system can be thought of as concurrent composition of three components as depicted in the diagram below:



Each component is a process, an active entity implemented as a thread, that operates concurrently and synchronises with other processes through specific events (communication interface between the components). The parallel composition therefore will be an alphabetized parallel in which synchronisation happens over certain events. You only need to define a minimal set of events for each processes so that the observable behaviour of the entire system can be completely characterized.

You can add as much detail to the implementation for each of the component. In refining the design by adding details your goal should be such that the components closely model a real parking application.

As a minimum, each component should itself be designed as a concurrent composition of sub-processes as follows:

- the Car Park process should be designed as three processes: arrival, departure and controller process that keeps track of the status of the space of the car-park.
- The e-ticket process should be modelled along the mail-tool process presented in the lecture notes.

- The level of detail in the Booking component is open ended and left to your imagination and creativity!!! I will judge it more on the concurrency concepts used in its implementation than on its aesthetics !

What you have to do :

Implement a program for the online car park booking problem in any concurrent programming language of your choice eg., JCSP, PyCSP, Jibu etc which is based on CSP. You can implement and test each component of the design shown above separately so that the entire system can be developed incrementally.

Marking Scheme (marked out of 100):

30 % Online booking component implementation and testing. 5% for the GUI so don't put too much effort into it. Go for it if you want to unleash your passion for GUIs !
30 % e-ticket component implementation and testing.
20 % car park component implementation and testing. 5% of it is for incorporating any additional features.
10% For integrating the full system and testing.
10 % Write-up

What to hand-in :

A writeup with program listing and screen shots of test examples.

2. Concurrency Concepts in Java/C#:

Look up literature on lock-free concurrent data structures. Investigate the support for "lock-free" operations in java.util.concurrent package (or similar in C#) with the following implementations:

1. A counter example where a number of threads update a counter value.
2. The Car park controller implemented with lock-free data structures instead of communicating channels between arrive and depart processes. Verify with test cases that the booking system operates correctly with this modification.

Based on your investigations, write a short technical report (no more than 2 pages) discussing the following aspects:

- Lock-free versus Wait-free data structures
- Advantages of lock-free/wait-free concurrency over standard concurrency using synchronized methods in Java/C#
- Include the code listing and test cases for the lock-free investigations above.

Marking Scheme (out of 100):

60% Implementation and testing
40 % technical report

About the Individual Write-up :

The write-up should be presented in the form of a technical report.

Your report should have the following structure:

- Title and Abstract
- Section 1:
  An introductory section in which you should briefly describe CSP and other concurrency models. Issues in concurrent systems i.e. deadlocks, livelocks, race conditions, starvation. The monitor model in modern concurrent programming languages like Java/C# and its limitations. And a bit of description on the application itself.
- Section 2:
  Modelling the car parking problem in the CSP formalism
- Section 3:
  Implementing the solution in JCSP, PyCSP, Jibu etc.,. Test cases to validate the implementation.
- Section 4:
  Some discussion on how you are confident on the correctness of your solution in terms of the trace model.
- Section 5:
  write-up on “lock-free/wait-free” Concurrency Concepts in Java/C#
- Section 6:
  Conclusions on the design and implementation and general concepts in this coursework.
- Appendix I
  Code listing