Homework 5

Please send your solutions via email to the instructor until January 13, 2011.

Problem 1 Scala Actors (5 Points)

Scala Actors are part of the standard library of the Scala language. For instructions on how to install the Scala language distribution see [http://www.scala-lang.org/](http://www.scala-lang.org/) If you use Linux then your Linux distribution will most likely provide corresponding packages.

(1) News Feed Distribution Service (2 Points)

Implement a news feed distribution service. The service consists of a central server and two types of clients: consumers and producers. The central server provides a selection of news topics to which consumer clients can subscribe. The producers can send news for a specific topic to the central server, which then distributes the news to the consumers that are subscribed for the corresponding topic. Producers also have to register with the central server before they can send their news. In particular, they have to inform the server for which topics they provide news. Design a protocol for the communication between the central server and the clients. Implement your service using Scala actors.

(2) Parallel shortest path computation (3 Points)

Design a parallel version of Floyd’s all-pairs shortest path algorithm (or find one on the web) and implement it using Scala actors.

Problem 2 Concurrent ML (5 + 2 Points)

Concurrent ML is part of the SML/NJ distribution. For instructions on how to install the SML/NJ distribution see the project web page [http://www.smlnj.org/](http://www.smlnj.org/) If you use Linux then your Linux distribution will most likely provide corresponding packages.

(1) Recursive Data Types (3 Points)

Similar to the implementation of reference cells that you have seen in class, one can use Concurrent ML to implement recursive data types. Implement a structure CList for a list data type with the following signature:

```plaintext
signature CList =

  sig
    type 'a clist
    val cnil : 'a clist
    val cons : 'a -> 'a clist -> 'a clist
    val tl : 'a clist -> 'a clist
    val hd : 'a clist -> 'a
    val map : ('a -> 'b) -> 'a clist -> 'b clist
  end
```

The semantics of the operations is defined by the following reference implementation that uses the inbuilt SML lists (the module List is part of the SML standard library).

```sml
structure CList : CLIST =
  struct
    type 'a clist = 'a list
    val cnil = nil
    fun cons x xs = x :: xs
    val tl = List.tl
    val hd = List.hd
    val map = List.map
  end
```

Your implementation should not use the inbuilt lists, respectively, the recursive data types provided by SML.

(2) Fixed-point Operator (2 + 2 Points)

The goal of this exercise is to demonstrate the computational power of the communication primitives provided by CML.

A fixed-point operator `fix` is a function that, given a functional `f` of type

\[(\text{'a} \to \text{'b}) \to \text{'a} \to \text{'b}\]

computes a function of type `\text{'a} \to \text{'b}` that is the least fixed-point of `f`. As an example, consider the functional `factFun` for the factorial function:

```sml
fun factFun (f : int -> int) (x : int) : int =
  if x = 0 then 1 else x * f (x-1)
```

Assuming the function `fact : int -> int` computes the factorial function then the function `fact'` defined as:

```sml
val fact' : int -> int = factFun fact
```

also computes the factorial function. That is, `fact` is a fixed-point of `factFun`. A fixed-point operator allows one to compute `fact` from `factFun`:

```sml
val fact = fix factFun
```

Thus, using the operator `fix` one can define recursive functions without using explicit recursion. A simple implementation of a fixed-point operator using the inbuilt recursion mechanism of SML is as follows:

```sml
fun fix f = f (fn x => fix f x)
```

Use CML to implement a fixed-point operator without using the inbuilt recursion provided by SML (i.e., it should be possible to syntactically transform your solution into one that does not use any `fun` or `val rec` declarations). **Hint:** you only need the operations `spawn`, `channel`, `send`, and `recv`. You may ask the instructor for additional hints on how to solve this exercise and you get 2 bonus points if you solve it without additional hints.