Erlang Project

Dr. Joseph Kehoe

January 13, 2022

1 Overview

Create a set of distributed communicating processes that communicate using the RIP protocol and compute primes.

- Each object will have a unique human readable dentifier called the **nickname**. Each nickname will map to an Erlang Process ID (PID);
- Each process will have between one and three direct connections to other processes. These are known as immediate neighbours. Communication between immediate neighbours uses the standard Erlang PIDs;
- Each process will only store/know the PIDs of its immediate neighbours. All other communication will use only the **nickname**;
- If a process needs to send a message to another process that is not an immediate neighbour then it must use the basic Routing Information Protocol (RIP)[1] to do so;
- Each process can respond to two messages:
    1. computeNthPrime;
    2. receiveAnswer.

2 Message Formats and Algorithms

2.1 computeNthPrime

This message consists of a tuple containing:
{computeNthPrime, N, DestinationNickname, SenderNickname, Hops}

computeNthPrime An atom identifying the message type;

[1] `https://en.wikipedia.org/wiki/Routing_Information_Protocol`

N A variable containing the index of the prime number we want computed. E.g. if N=5 then we compute the 5^{th} prime number (i.e. 11);

DestinationNickname The nickname of the process we are sending the message to;

SenderNickname The nickname of the sending process. The computed prime is send to this process in the receiveAnswer message;

Hops The number of processes that have forwarded the message so far (including the source process);

1. If the number of Hops is 15 or greater then we discard the message as unreachable;
2. If the DestinationNickname is not ours then we route the message to its final destination (using our local routing lookup table[2]);
3. If the destination nickname matches ours then we compute the N^{th} prime and send the appropriate **receiveAnswer** message back to the source.

2.2 receiveAnswer

This message consists of a tuple containing:
{receiveAnswer, N, M, DestinationNickname, SenderNickname, Hops}

receiveAnswer An atom that identifies the message type;

N A variable containing the index of the computed prime number.

M A variable containing the correct prime E.g. if N=5 then M=11;

DestinationNickname Th immediate neighbour list will contain the entries: [jane,PID1,phil,PID2, maxine, PID3] 3.2 Routing Table (RT) A list that maps nicknames to immediate neighbours. E.g. If 1. To send a message to the process fred we need to route it through jane and it is 5 hops away from us; 2. To send a message to the process bert we neee nickname of the destination process of the message;

SenderNickname The nickname of the process sending the answer;

Hops The number of processes that have forwarded the message so far(including the source process);

[2]See Process State

1. If the number of Hops is 15 or greater then we discard the message as unreachable;
2. If the DestinationNickname is not ours then we route the message to its final destination (using our local routing lookup table);
3. If we are the destination then we print the answer on the screen.

3 Administrative Functions

3.1 Node Startup

Nodes are launched and initialised with the call:
launchNode(Nickname) where:

Nickname is a string that will act as the process Nickname.

lauchNode returns the Process ID of the spawned process.

3.2 Node Connections

Initial connections between nodes are throught the call:
connectNode(NicknameOne,PidOne,NicknameTwo,PidTwo) where:

NicknameOne is the Nickname of the process with process ID PidOne;

PidOne is the Process ID of the node with Nickname NicknameOne;

NicknameTwo is the Nickname of the process with process ID PidTwo;

PidTwo is the Process ID of the node with Nickname NicknameTwo;

connectNode returns true is the two nodes are successfully connected as *immediate* neighbours.

3.3 Displaying Route Tables

A nodes routing table can be printed by calling:
printTable(Pid)
This will be used when testing the network setup to make sure that RIP is operating correctly.

4 Process State

Each process will need to hold the following state:

4.1 Immediate Neighbour List (INL)

A list containing structs mapping nicknames to PIDs. This will contain at most three elements and at minimum one. These are the only Erlang PIDs that the process has access to. E.g. if we are immediate neighbours with the processes nicknamed jane, phil and maxine then the immediate neighbour list will contain the entries: [{jane,PID1},{phil,PID2}, {maxine, PID3}]

4.2 Routing Table (RT)

A list that maps nicknames to immediate neighbours. E.g. If

1. To send a message to the process **fred** we need to route it through **jane** and it is 5 hops away from us;
2. To send a message to the process **bert** we need to route it through **phil** and it is 2 hops away from us;
3. To send a message to the process **mary** we need to route it through **jane** and it is 9 hops away from us.

Then the RT will contain the entries [{fred,jane,5},bert,phil,2},{mary,jane,9}].

5 Hints

- RIP will require other messages to be implemented (for constructing and updating the RT etc.). Once the initial immediate connections are made then the RIP protocol implementation should update and produce correct Routing Tables for each node.
- Other functions not listed here will be required e.g. for testing purposes
- An iterative refinement approach to implementation is recommended.

6 Due Date

The project must be submitted to Blackboard by the 18th of March 2021. All code must be fully documented.