Distributed Algorithms 347

Labs 3 and 4 – Coursework 1

01 Labs 3 and 4 will be available for first coursework. There will be a 2nd coursework later in the term.

02 The aim of this coursework is to combine together much of the work done in the previous labs and the lectures. We recommend that you finish lab 2 before starting on the coursework.

03 You can work and submit either individually or jointly with one classmate.

04 You are strongly encouraged to start the coursework from scratch - you will learn more by doing this and probably come up with cleaner code.

05 Don't assume that the code that has been provided in the lab or shown in the lectures is suitable for this coursework in the form given, you will adapt it to get it to work.

06 Again, it's important that you have an incremental and experimental approach to developing the systems in this coursework

07 Write comments to explain what less obvious code is doing - don't over-comment. Don't use super long names for atoms, variables, functions etc - unless super long names help you to understand what's going on. *Write comments to help you to revise the material in the future*. If you code does not work or only partially works you must explain what's not working.

08 Do comment in a separate note file on the tests you conduct and they suggest to you.

09 All your source files must include a comment at the top of the files with your name(s) and login(s) e.g.,

```
%%% Mary Jones (mj16) and Peter Smith (ps16)
```

10 Source files without your name(s) and login(s) **will not be marked**.

11 The deadline for submission is **Wednesday 15th February 2017.**

12 Submit the directory of your code, output files, build and execute files (e.g. Makefiles), readme file, other information (text or pdf only) as single *zip* file on Cate.

13 It should be possible for us to run systems using your instruction.

14 Your submitted files must also print correctly on DoC printers so check first - you will not be given an opportunity to resubmit if a file does not print correctly.

15 Use Piazza if you have questions about the specification or general questions. **Do not post your solutions or share with others.** Email me directly if you have a question about your solution.

16 The most up-to-date version of the coursework will be on the course webpage.

System1 – Erlang Broadcast

17 Create a system (`system1`) with N=5 fully connected processes . The structure is

```
system1
  process1, .., processN
```

18 Each process should broadcast `Max_messages` and stop when `Timeout` (in milliseconds) is reached. After spawning and binding the processes, `system1` should sent a message `{task1, start, Max_messages, Timeout}` to each process for them to start the task. For the message `{task1, start, 1000, 3000}` the output should be:

```
   1: {1000,1000} {1000,1000} {1000,1000} {1000,1000} {1000,1000}
   4: {1000,1000} {1000,1000} {1000,1000} {1000,1000} {1000,1000}
   3: {1000,1000} {1000,1000} {1000,1000} {1000,1000} {1000,1000}
   5: {1000,1000} {1000,1000} {1000,1000} {1000,1000} {1000,1000}
   2: {1000,1000} {1000,1000} {1000,1000} {1000,1000} {1000,1000}
```

19 Each process outputs one line after the timeout is reached. The first number on an output line is the process number P (from 1 to N), the rest are pairs of values, one for each process Q (1 to N) where the first number of the pair is the number of messages broadcast from Process P to Process Q, and the second number is the number of messages received by process P from process Q. The order of the lines does not matter, since the system is non-deterministic. The order of the pairs is Q=1..N

20 If `Max_messages` is 0, processes should keep broadcasting and receiving messages until the `Timeout` value is reached.

21 The challenge of this task is not in the message-passing involved, rather it's how accumulate the output required and timeout the task.

22 Your code shouldn't broadcast all the messages and then receive them. Rather it should be doing both non-deterministically. The value 0 and the atom `infinity` have special meaning in the `after` clause of a `receive`. You may find that useful to terminate the task.

23 If you decide to use maps, it is recommended that you stick to calls to functions in the `maps` module like `get`, `update`, `from_list` rather use native syntax, which gets "ugly" when pattern matching.

24 Other Erlang functions you may find useful are `timer:send_after`, `timer:sleep`, `io_lib:format`, `lists:flatten`, `erlang:halt`

25 Submission: show and comment on the output of `system1` for message `{task1, start, 1000, 3000}` followed by message `{task1, start, 0, 3000}`.

26 Optional: distribute and run `system1` on a Docker network of containers, similarly for other systems below.

System2 – PL Broadcast

27 The aim of `system2` is to refactor `system1` to use perfect p2p links (PL) giving the following hierarchy:

```
9   system2
10    process1
11      PL, app
12    ...
13    processN
14      PL, app
```

28 One way to connect PL components to each other is for `process` to send the process-id of its PL component to `system2` and for `system2` to send a suitable `bind` message to each PL component. `process` can return once its created its components.

29 Once created and bound to its PL component `app` takes on the role of `process` in `system1`. However `app` communicates with other processes using its PL component e.g. receiving `pl_deliver` messages from `system2` (to start `task1`), and from other processes to carry out the `task1`.

30 You can use 0 as the number of the `system2` process if you wish.

31 Submission: show and comment on the output of `system2` for 100 broadcasts with a 1 second timeout followed by an unlimited number of broadcasts (0 `Max_messages` value) with a 1 second timeout.

System3 – Best Effort Broadcast

32 The aim of `system3` is to refactor `system2` to use best effort broadcast (BEB) components. The hierarchy of `system3` will typically be:

```
15  system3
16    process1
17      PL, BEB, app
18    ...
19    processN
20      PL, BEB, app
```

33 Submission: show and comment on the output of `system3` for 100 broadcasts with a 1 second timeout followed by an unlimited number of broadcasts (0 `Max_messages` value) with a 1 second timeout.

System4 – Unreliable Message Sending

34 The aim of `system4` is to refactor `system3` to add a reliability parameter to the PL components to simulate unreliable message passing.

35 Write and use `lossyp2plinks` to add an integer reliabijlity percentage from 0 to 100. 100 sends all messages (100% reliable). 0 drops all messages (0% reliable). 20 sends ~20% of the messages - use a suitable random number test to decide whether to send a message or not.

36 Test you `system4` with different reliability values.

37 Submission: show and comment on the output of `system4` for 100 broadcasts with a 1 second timeout followed by an unlimited number of broadcasts (0 `Max_messages` value) with a 1 second timeout, using reliability values of 100, 50, and 0 respectively.

System5 – Faulty Process

38 The aim of `system5` is to refactor `system4` and make process 3 terminate after 5 milliseconds (choose a different termination timeout if it helps).

39 The function `exit` can be used to terminate an Erlang process.

40 Submission: show and comment on the output of `system5` for 100 broadcasts with a 1 second timeout followed by an unlimited number of broadcasts (0 `Max_messages` value) with a 1 second timeout, using a reliability value of 100 in both cases.

System6 – Eager Reliable Broadcast

41 The aim of `system6` is to refactor `system5` to use eager reliable broadcast (BEB) components. The hierarchy of `system6` will typically be:

```
21 system6
22   process1
23     PL, BEB, RB, app
24   ...
25   processN
26     PL, BEB, RB, app
```

42 The aim of `system6` is to refactor `system4` and make process 3 terminate after 5 milliseconds.

43 The function `exit` can be used to terminate an Erlang process.

44 Submission: show and comment on the output of `system6` for one or more sets of system parameters, e.g. number of messages, timeout for completion of task1, reliability value, timeout to use to terminate process 3, etc. The aim is to demonstrate the operation of the system and your understanding of how it works. You can suggest other parameters/components you would add to test this or systems 1 to 5.

45 Optional: Repeat for lazy reliable broadcast.

46 Optional: Add an underlying multi-hop network and route messages over it.

47 ---------- That's all folks ----------