CS60002: Distributed Systems
Spring 2018
Assignment – 1

Due: January 17, 2018 in class

1. Write a distributed algorithm for constructing a spanning tree rooted at a specific node in an asynchronous, reliable system. The root node will initiate the algorithm. At the termination of the algorithm, each node should know its parent and children set in the spanning tree. In addition, the root node should know that the spanning tree has been constructed (i.e., the algorithm has terminated).

2. Repeat the above to construct a BFS spanning tree.

Guidelines for how the algorithm should be written (gives you an idea with an example. Your answer need not match it exactly, but should have all necessary information)

1. Specify the model/assumptions
2. Specify the type and content of the messages that you will use, with 1-line description of each field of the message
3. Write the code for each node (if a group of nodes execute the same code, write them only once)
    a. Write the variables at each node, with a 1-line description of what it is to be used for. Exact data structures are not needed, you are writing psuedocode.
    b. Write an initialization section showing the initial values of the variable, if any
    c. Write a handler psuedocode (a subroutine) for each event that can happen in the system. The handler will be called when that event happens. How is it called is not your concern. Events that you will need for the assignments are wakeup/start/init (to start the algorithm), send of each type of message, receive of each type of message. Usually, first write the wakeup (if needed) and the receive handlers, think if send handlers are needed separately or not as in many cases sends are done in response to receives.

Example: Algorithm for flooding. Any node can start a flooding message

Model/Assumption:
- Asynchronous system
- Reliable communication
- Arbitrary topology
- Each node has unique id
- Each node knows its neighbour set N

Messages:
- M(sender, id, data)
    - sender: original sender of the message
    - id: unique id of the message corresponding to that sender
    - data: any other data that is to be flooded

Program for node i

Variables:

- msg_cnt: no. of flooding messages sent by this node
- L: list of messages seen so far

Initialization:

- msg_cnt = 0; // no messages sent at the beginning
- L = empty

Handlers:

On startup (invoked whenever i wants to start flooding a message)

```
for each node j in N
        send M(i, msg_cnt, data) to j
msg_cnt++;
Add M to L
```

On receiving M(x, y, data) from node j: (this can be a message initiated by i or others)

```
If M is in L        // already seen, nothing to do
        Drop M
Else                // seen for the first time, so forward to all but sender
        Add M to L
        for each node k ≠ j in N
                send M(x, y, data) to k
```

Note that there is no other send handler needed as all cases where a node will send a message M is covered by the above handlers. So nodes just sit there waiting for an event and respond to it. Some events have external stimulus, for example a startup event can be invoked by an application wanting to flood something, which invokes a send event to send. A receive event is invoked by the system when it receives a message.