TI-TEDI Exercise: Broadcast based token passing algorithm for mutual exclusion

Contents

Introduction

In this exercise we are going to investigate a "Broadcast based token passing" algorithm for mutual exclusion in distributed systems.

Setup and assumptions

We have N fully connected nodes, which means that each node is connected to all the other nodes. More specifically we make these assumptions:

- **A.** Each node can directly send a message to any other node.
- **B.** Each node can broadcast (or multicast) a message and all the other nodes will receive it.
- **C.** The network is reliable so messages do not get lost and messages are delivered in (FIFO) order[1].
- **D.** We don't have a common clock, shared memory and we cannot assume anything about processing speed at the nodes, nor anything about message delivery delay - i.e. we're talking about an asynchronous distributed system.
- **E.** Processes do not get stuck in their critical region.

So our system might look like one of these models (see page 2):

[1] I.e. message delivery semantics matches that of Promela channels (lucky for us, eh?)

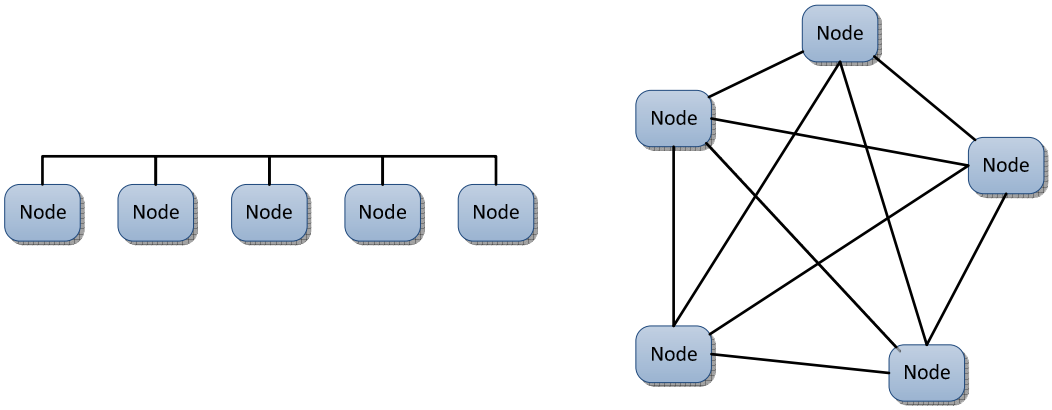

Figure 1. Different topologies

The algorithm

The algorithm works like this:

1. At any one time only one process holds the token (or the token is in transit from one process to another)
2. If a process wants to enter its CR (critical region)
    a. If it holds the token it simply enters the CR
    b. If it does not hold the token, it
        - Broadcasts a message REQUEST(I) (where I is the id of the requesting process) to all other nodes
        - Waits for a TOKEN message, indicating that it now holds the token
        - Enters its CR
3. A process that receives a REQUEST(I) message responds like this
    a. If it holds the token it sends a TOKEN message directly to process I
    b. If it does not hold the token, it does nothing

So, basically all nodes sees the REQUEST messages but only one process receives a TOKEN message (the one who gets the token).

A simple solution

The following listing shows a SPIN model that implements a first shot of an implementation of the "broadcast token passing algorithm for distributed systems" (you can also find the code in the file broadcast_simple.pml) for 3 nodes.

```
// Define's and ghost variables for checking mutex &
// starvation properties
#define mutex (critical <= 1)
byte critical = 0;
#define nostarve (incrit[0] && incrit[1] && incrit[2])
bool incrit[3];

// Only 2 types of messages, request token or taken granted
mtype {tokreq,tokgrant};
chan ch [3] = [3] of {mtype, byte};
```

```
proctype P(byte id)
{
    // Initialise stuff
    incrit[id] = false;
    bool holding = false;
    bool waiting = false;
    byte requester;
    byte pidx;
    end:
    do
    // Token request received
    :: ch[id] ? tokreq,requester;
    if
    :: holding;
       ch[requester] ! tokgrant,id;
       holding = false
    :: else -> skip
    fi
    // Token granted
    :: ch[id] ? tokgrant,_;
       holding = true;
       waiting = false;
       critical++; // Enter & leave CR
       critical--;
    // If there are no messages in queue and we don't have
    // the token (and we haven't asked for it yet), request token
    :: !holding && !waiting;
       // Broadcast request for token
       for (pidx : 0 .. 2) {
         if
         :: (pidx != id); // Don't send request to ourselves
            ch[pidx] ! tokreq,i
         :: else -> skip
         fi
       }
       waiting = true
    od;
}

init
{
    // Send a single token into the system
    ch[0] ! tokgrant,0;
    // and start the processes.
    atomic {
        run P(0);
        run P(1);
        run P(2);
    }
}
```

```
// LTL Formulas - uncomment as needed
// * Check for mutual exclusion
// ltl{[]mutex}
// * Check for starvation
// ltl{[]<>nostarve}
```

Exercise 1 – Look at the simple SPIN model

Look at the SPIN in the listing above and make sure that you understand how it works - do you agree that it models the algorithm as described?

Exercise 2 – Checking the simple model

Try to run the simple model and see what happens if you try to verify the two properties of:

1. Mutual exclusion
2. Freedom of starvation.

Note: *To run the model you should increase maximum steps and maximum search depth like this:*

- *Maximum steps = 2500*
- *Maximum search depth = 100000*

Exercise 3 - Spotting the problem

By now you should have discovered that there is a problem with the simple model. Find the problem and describe the set of circumstances that create the problem.

Hint: Use the options that allow you to “fine-tune” the output:

1. Skip variables and statements that are less important.
2. Set the width of statements and variables in such a way that the length of the lines in the output becomes as small as possible but still readable.
3. Copy-paste the output into a program that allows you see the outpu with and without line-wrapping[2].

In this way you can cut down the size of output quite dramatically which makes it much easier to read.

Exercise 4 - Removing the problem

Once you feel confident that you have understood the problem, try to come up with a solution. You should probably do this as a 2-step process:

1. Describe your solution and try it out on a piece of paper - see if it can handle the specific problem that trips up the simple model.
2. Try to modify the simple model so as to incorporate your solution.

[2] Notepad++ is free and VERY good at stuff like this. Just Google for it.