The University of Melbourne
School of Computing and Information Systems
COMP90020 Distributed Algorithms

Tutorial Week 5: Mutual Exclusion and Failure Detection

Notes

In this tutorial we will look at mutual exclusion and briefly at failure detectors. Both of these topics have much more to offer. In particular, we will only examine mutual exclusion using message passing and won't look at algorithms that use shared memory.

If you're interested to learn more broadly on these topics, a great place to start is Lamport's bakery algorithm and further continue onto queue, CLH and MCS locks.

Exercises

18. Unreliable - yes, all that changes is that there may be a higher false positive rate (detecting process failures when instead its due to unreliable delivery). Reliable - No! We cannot implement a reliable failure detector on either channel type without an upper bound on message delay i.e. a sync system.

19. Mututal exclusion is a safety property, it must be met in order to ensure correctness - where correctness is met as at most one process can be in the CS at any one time. So in other words, something breaking should never happen. Liveness properties say that something good (progress) should happen in some finite time. Starvation indicates that a process cannot progress, therefore starvation-freeness must be a liveness property. An interesting follow up - under the definitions presented, deadlock-freeness is also a liveness properties however, in some textbooks it is defined separately as a safety condition.

20. Consider the following events consisting of three processes: A, B and the central server C

- A requests entry to CS (sending a message to C)
- A sends a msg to B (could be anything)
- B receives A's message
- B then requests entry to CS (sending a message to C)
- C receives B's request first and responds with the token
- B receives the token and enters CS
- C receives A's request and queues it.

Clearly, in this set of events - there is a HB relation on A requesting entry and B requesting entry. This ordering was not honored at C as B was granted the token instead of A. Therefore ME3 has been violated.

21. We can propose a similar example to the question above, consider three processes A,B,C arranged in a ring such that A has a directed edge to B, B has a directed edge to C and C has a directed edge to A. Assume the token starts at C

- B requests entry to its CS (sends a message to C) and enters wanted state
- B sends a msg to A (could be anything - this message exists outside of the abstract topology of directed edges for the ring algorithm)
- A receives Bs message
- A requests entry to its CS (sends a message to B) and enters wanted state
- C exits its CS and passes the token to A.
- A receives the token and enters its CS

As before, we have violated ME3 as there is a HB relation on B requesting entry and A requesting entry that is not honored.

22. This problem is akin to a constraints question and not a likely candidate for any timed test.

$$\begin{aligned}
S1 &= \{1,2,3,4\} \\
S2 &= \{2,5,8,11\} \\
S3 &= \{3,6,8,13\} \\
S4 &= \{4,6,10,11\} \\
S5 &= \{1,5,6,7\} \\
S6 &= \{2,6,9,12\} \\
S7 &= \{2,7,10,13\} \\
S8 &= \{3,6,8,13\} \\
S9 &= \{3,7,9,11\} \\
S10 &= \{3,5,10,12\} \\
S11 &= \{1,11,12,13\} \\
S12 &= \{4,7,8,12\} \\
S13 &= \{4,5,9,13\}
\end{aligned}$$

23. The L-exclusion problem can be considered a general case of the mutual exclusion problem. Therefore in normal circumstances, when we wish at most L=1 process to enter its CS at any time we require $N-1$ acks from processes (not including itself). We can generalize this to $N-L$ acks and note that ME1-3 still hold. See the excalidraw for some examples. There were also some solutions that suggested use of tokens that acted as leases - this solution appeared viable for ME1,ME2 however couldn't satisfy ME3.

24. Consider the following steps for each process i:

(A) Process i sends a time stamped request to each neighbor

(B) A process sends an ack to a neighbor j when it is not in its CS, or its own timestamp is greater (it wants the resource)nd it has previously requested the resource.

If the receiving process is in its CS then it sends the ack to each requesting process when it exits its own CS.

(C) A process enters its CS when it receives an ack from each of it neighbors.