Learning and Skills Outcomes

- **Learning Outcomes:**
  - **Deeper understanding of ‘happened before’ relation in a distributed system**
  - **Importance of logical clocks and their use**
  - **Significance of clock error in synchronized physical clocks**
  - **Merits/shortcomings of logical/physical clocks**
  - **Foundations for Coursework 2**
- **Skills Outcomes:**
  - **Analytical**
  - **Presentational (writing)**

Coursework 2: Designing a Total Order (TO) service

CONTEXT:

- Consider a distributed system of *n*, *n* > 3, processes: P1, P2, …, P*n*
- A process can generate an (application-related) message *m* and send *m* to an arbitrary subset of processes in the system
  - There is no assumption that *m* must only be sent to every other process in the system
- Each process has an instance of TO service implementation in its local node.
  - Let us call the instance of TO for process Pi , 1 ≤ i ≤ *n*, TO_i.
- When, say, P1 sends m to P2 and P3, it gives the message to local TO_1 which, in turn, timestamps m with a logical cock and sends m only to the indicated destinations.
- When m reaches the host node of, say, P2, the TO_2
  - Receives the message (on P2's behalf),
  - Puts it in a local queue as per the timestamp, and
  - Deduces the right moment to hand-over m to P2
- TO_i handing over a received m to local Pi is called delivering m
  - Delivering m is an irreversible operation;
  - TO_i cannot
  - Deliver m to Pi, change its mind, un-deliver m and then re-deliver m.

Coursework 2: Specification

- **The Total Order service must satisfy the following two conditions in delivering the received messages to processes:**

  (C1) Say messages m and m' are sent to process Pi. If sending of m *happened before* sending of m', then the delivery of m to Pi (by TO_i) must happen before the delivery of m' to p.

  (respects "happened before")

  (C2) Say messages m and m' have common destinations of, say, Pi and Pj. The deliveries of m and m' to Pi and Pj (by TO_i and TO_j respectively) must occur in an identical order:

  either the delivery of m precedes the delivery of m' at both p and q or delivery of m' precedes delivery of m at both p and q

- **Service should NOT**
  - **Require a TO_i to receive and then either discard or forward a message not destined for itself or Pi**
    - Every message or ack that TO_i receives must carry some useful info for implementing the service

Assumptions

- **(A1) There is NO access to synchronised physical clocks;**
- **(A2) A message by a source TO is eventually received at a destination TO, but no known bound on transmission delays;**
- **(A3) Between any two TO_i and TO_j, messages sent by TO_i to TO_j are received in the sent order;**
- **(A4) Sending of a message to one or more destinations may be considered as a single event at the sending end (so, copies of a message sent to multiple destinations have the same timestamp), and**
- **(A5) Processes are uniquely ordered; this ordering is known to all TO processes.**

Coursework 2 : Requirement

Required: A distributed design for the total order service.

(12 Marks, 6 Pages max)

- You **must describe your design** in terms of how a TO_i carries out its
  - Message sending task
  - Message receiving task, and
  - Message delivery task
- You **must also provide** an **informal** argument on how your design meets its specification, i.e., C1 and C2.
- Use example to expand on design aspects and to provide correctness arguments.
  - '**Informal**' means example based and as arguments by contradiction
  - To keep within the page limit, you are advised to think and select such examples that explain multiple cases/aspects
- Correctness of design and clarity of explanation are the only criteria for marking this coursework
  - A **trivial design** meets C1 and C2: do not deliver any received message; delivery task simply discards every received m; this trivial design **will receive zero** marks
- Submission and Mark scheme criteria are as in Coursework 1

Coursework 2 : Hints

A TO_i has a queue of received messages that are due for local Pi.

The Transparency T10 ensures that the messages, when ordered as per the logical clock timestamps, enforce a total order – captured in conditions C1 and C2 of the specification

The problem for TO_i is this:

Say, m with C(m)= 5 is at the head of the queue. It has to know what it should observe to conclude that it can NEVER receive another m' with C(m') ≤ 5? Because, if TO_i delivers m to Pi (for processing) and then receives m' with C(m') = 4 arriving, it would have failed Pi. Why?

Both m and m' are destined for Pi, C(m') < C(m), and m' is before m in the total order; but m is delivered before m'!

So, the design challenge here is: *What should each TO_i should do, in addition to (i) implementing the local logical clock (ii) sending and receiving messages on behalf of Pi (iii) and queuing the received messages?*

•Hints for this critical part can be found in the mutual exclusion problem we discussed. It requires TO_i using acks to make the necessary deduction.

Learning and Skills Outcomes

- Learning Outcomes:
  - deeper understanding of design issues in accomplishing total order on messages
  - Relating mutual exclusion as a total order issue in distributed systems
  - Foundations for chapter 4: Global system state
- Skills Outcomes:
  - Analytical
  - Application of facts for system design
  - Use of examples to demonstrate correctness
  - Presentational (writing)

Submission & Mark Scheme

- **Submission Format**:
  - Word or PDF,
  - Pages refer to A4 size, 1- or 2- columned
  - Page limit includes figures, appendices and any references.
  - Submission in NESS by the prescribed deadline
  - You do not have to refer to transparencies in lecture handouts
- **General Mark Scheme** (for each sub-question):
  - 50% for correctness and 50% for clarity of write-up given that solutions are correct
  - Totally incorrect answers receive no marks however well presented
- Notes:
  - There is no one correct answer, even though there is certainly one most common and correct answer.
  - Your answer will be judged based only on its own merit – that is, how correct it is and how well it is presented
  - It will NOT be judged based on how close it is to the common correct answer
  - Each submission will be marked by the module leader, not by demonstrators.