9.7 Distributed systems fundamentals

Name: _____ Class: _____ Date: _____ **Total: 24 marks**

KEY WORDS Lamport clock • idempotence

Objective

Compute Lamport clocks and design idempotent retries.

1 Key knowledge and worked examples

■ The clock

Send: attach $C + 1$. Receive: $C = \max(C, \text{received}) + 1$ ✓. Course trace: send at 1, receive at 2, send at 3, receive at 4 ✓.

■ Worked example: the one-direction rule

If a happens-before b then $C(a) < C(b)$; the converse is false ✓. A timeout means unknown, not failed ✓.

2 Practice

2.1 P1 at $C = 5$ sends to P0 at $C = 2$. What is P0's next C? [3]

2.2 Can two events share a Lamport value? [3]

2.3 What does idempotence guarantee for a retried operation? [3]

3 Exam-style questions

3.1 P0 (C = 3) and P1 (C = 7) exchange: P0 sends m_1; P1 receives and immediately sends m_2; P0 receives m_2; P0 then sends m_3.

Compute the Lamport value attached to each message and each process's counter after every event (list the sequence). State which pairs are ordered by happens-before and which are concurrent. State why wall clocks could not have produced a guaranteed order. [8]

3.2 A payment service exposes `deduct(user, amount, id)` and clients retry on timeout.

State the double-charge risk and when it fires. Give the idempotent design (key plus ledger) and what the retry returns. State what a timeout proves and what the client should do. Give the rule for choosing which operations must be idempotent. [7]

Solutions

2.1

- $C = \max(2, 5) + 1 = 6$ ✓.

2.2

- Yes – at different processes; equal clocks imply nothing about order ✓.

2.3

- Executing it more than once has the same effect as once ✓.

3.1

- m_1 sent at $C = 4$ (P0: $3 \to 4$) ✓.
- P1 receives: $C = \max(7, 4) + 1 = 8$ ✓.
- m_2 sent at $C = 9$ (P1: $8 \to 9$) ✓.
- P0 receives: $C = \max(4, 9) + 1 = 10$; m_3 sent at 11 ✓.
- Ordered: send(m_1) before receive(m_1), before send(m_2), before receive(m_2), before send(m_3) – a causal chain ✓.
- Concurrent: none in this trace – every event lies on one chain; concurrent events would share no arrow path ✓.
- Wall clocks skew without bound across machines: a later event can carry an earlier timestamp, so no guaranteed order ✓.

3.2

- Risk: the request lands, the reply is lost, the client retries – two deductions fire ✓.
- Design: idempotency key `id` plus a ledger – the first execution records `id` as done; the retry looks it up and returns the original result without re-executing ✓.
- A timeout proves nothing about execution – the client retries, and safely, because the operation is idempotent ✓.
- Rule: every operation a client may retry (any network call) must be idempotent ✓.