Sorted Queue of Messages

no duplicates

int 64\_t

\*0 microsenconds - 0 ~ 999, 999

-need machine ID

variables a, b, c, d

int 64\_t b = a \*1,000,000;

int 64\_t c = b + #usecs

int 64\_t d = (c\*10) + machineID

=> change mailheader

%llu

Distributed Mutual Exclusion

Token Ring Algorithm

Requirements:

1) Reliable communication

2) Globally unique ID

3) Members know ring order of all

For each C.S., there is a token msg

it contains the C.S. ID

To enter a C.S., the member must have the token

hold the msg while in the C.S.

upon C.S. exit, forward the msg to next member

If a member receives a token & does not want to enter the C.S. - forward token

+Correct, Fair - Single point of failure( system went down)

Election Algorithm

To be able to recover from a critical failure in a D.S.

Requirements:

1) Guaranteed method for deciding the critical member is dead

2) A reliable method for selecting a new critical member

3) These methods must work even if multiple.

Members are doing 1&2 about the same time

Step 1: Determine the C.M. is dead

Step 2: Select a new C.M. All members must agree

Bully Algorithm

Send an Election msg to all members w/ an ID larger than mine

Upon receipt of an Election msg, respond w/ an OK msg

Repeat until 1 member wins

no one to send to

all above me are dead

Winner send s a msg to all others - I’m new C.M.

Token-ring: create a token

Worst Case: All members decide C.M. is dead

\*Solution: add timestamps

+Correct

-Lots of msg

Ring Algorithm

Uses a “ring” architecture like Token Ring

Initiating member sends an election msg to my neighbor - I add my member ID

Each member receiving the msg appends their ID

Eventually msg returns to the initiator, change the msg ‘Completed Election’. Send msg to neighbour

Initiator discards msg when it return

+Correct

+O(N) msgs

+Works even w/ members dying during election

Election Algorithms are 1 aspect of Fault Tolerance

Define: A system that can perform in the presence of faults

Faults can product error in a system

Errors can use partial, or total failure in a system

Failed System: A system that cannot meet its obligations

Goal for Dist. Systems

Build them so they can recover from partial failure => Dependable System

4 Measures of Dependable Systems

1. Availability: Sytem ready for use measure: %up time

2. Reliability: How long can the system run continuously before failure, measure: mean time between failure

3. Maintainability: Relates to a partial failure - how much to fix the system

4. Safety - Cost to fix the damage outside the system

How to build Dependable System?

Control faults

Prevent

Remove

Forecast/Plan

Goal: Fault Tolerance

Must operate (maybe in reduced way) in the presence of faults

3 Types of Faults

Transient: Occur once, disappears, comes back “unexpectedly"

Intermittent: Occur, go away, come back, “regularly"

Permanent: Stay!

5 Types of Failures

Crash: System worked until failure

Omission: System does not respond to a request

Timing : System response is outside acceptable time range

Response: System response is wrong

Arbitrary: None of the above

Redundancy for controlling faults

Information: Add extra bits to data to detect/correct errors

Time: Perform an action/task, get the result, repeat more times - compare the results

Physical - more hardware