University of Dublin
Trinity College

Distributed Coordination

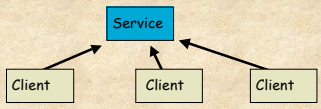
Distributed Systems 3

Distributed Coordination

Distributed processes often need to **coordinate** their activities

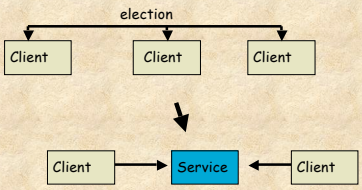
Two principle coordination models

- Dedicated Service



Distributed Coordination

- Coordinator chosen when needed via election from all processes



Distributed Mutual Exclusion

Two basic requirements

- Safety - At most, one process may execute in the critical section at a time
- Liveness - A process requesting entry to a critical section is eventually granted it (so long as any process executing in the critical section eventually leaves it)

One additional requirement

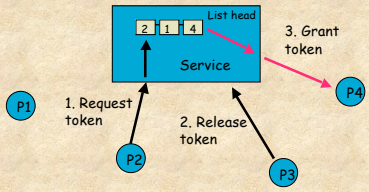
- Ordering - Entry to the critical section should be granted in 'happened-before' order

Central Server Algorithm

- To enter a critical section, a process sends a request messsage to the server and awaits a reply from it.
  - *reply → token for access*
- At time of request, if no other client has access token, server replies immediately, granting the token.
- If token, at time of request, is held by another process, server does not reply. Instead it queues the request.
- When client (who has token) exits the critical section it sends a message to the server, giving back the token.

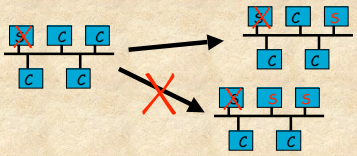
Central Server Algorithm

- If queue of waiting processes is not empty, server choses the oldest entry in queue, removes it from queue and replies to the corresponding process (giving it the token).



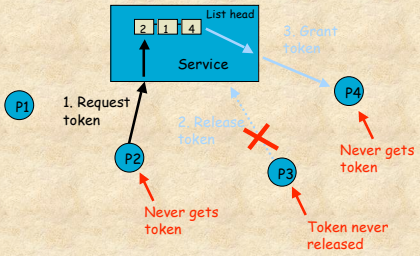
Disadvantages of CS Algorithm

- Server might become a bottleneck
- Server is critica point of failure
  - *Must provide means to select new server.*
  - *This server must be unique - requires a multicast.*



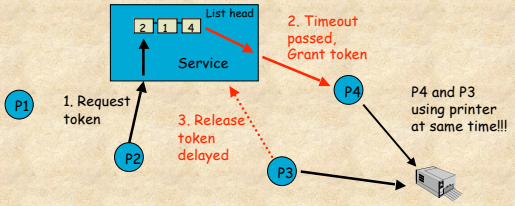
- Failure of client process holding token will cause other clients to be locked out. Will a timeout solve the problem?

Central Server Algorithm



Central Server Algorithm

Timeouts?



Distributed Algorithm

- Implements mutual exclusion based on distributed agreement, instead of using a central server, using **Logical Clocks**
- Processes that require access to critical section must multicast a request message and can only enter critical section when all other processes have replied
  - *i.e. only have token when all other processes reply*
- Algorithm assumes
  - *all processes know each other*
  - *all messages are eventually delivered*
  - *each process keeps a logical clock*
- Messages form: (Ti, Pi)

  Ti - senders timestamp Pi senders identifier

Distributed Algorithm

- Each process is in one of three states
  - *RELEASED - does not have or want the token*
  - *WANTED - wants but does not have the token*
  - *HELD - currently has the token*
- On initialisation
  - *state = RELEASED*
- To obtain token
  - *state = WANTED*
  - *Multicast request to all processes*
  - *T = request's timestamp*
  - *wait until received reply from all other processes*
  - *then state = HELD*

Distributed Algorithm

- On receipt of Request: (Ti,Pi) at Pj (i ≠ j)
  - *if state == HELD or (state == WANTED and (Tj,Pj) < (Ti, Pi))*

    then queue request from Pi - do not reply!

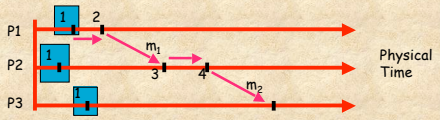
    else reply immediately to Pi
- To release token
  - *state = RELEASED*
  - *reply to any queued requests*
- If a process requests the token and the token is released everywhere else
  - *all processes will reply immediately to the requester*
  - *token is obtained*

Distributed Algorithm

- If token HELD at some process
  - *that process will not reply until it has finished with the token*
- If two or more processes request the token at the same time
  - *whichever process bears the lowest timestamp will be the first to obtain the token by collecting the necessary number of replies first*

Totally ordered logical clocks

- Logical clocks impose only a partial order on the set of all events.
- Some pairs of distinct events, generated by different processed, have numerically identical timestamps
- Need to order these events consistently for our algorithm to work



- Extend timstamp to 'Global Timestamp'

Totally ordered logical clocks

- If a is event occuring at Pa with logical timestamp Ta, we define global logical timestamp

  (Ta, Pa)

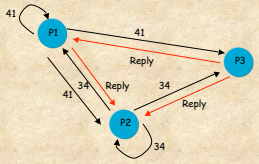
- Given (Tb, Pb), we define

  (Ta, Pa) < (Tb, Pb)

  iff either Ta < Tb or Ta == Tb and Pa < Pb

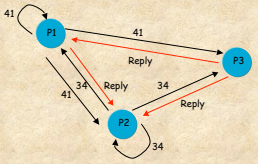
- i.e. arbitrary but repeatable and consistent ordering

Distributed Algorithm (cont)



- P3 not interested in the token
- P1 and P2 request token concurrently
- P1 timestamp of request is 41, P2 timestamp is 34
- When P3 recieves requests, it replies immediately
- When P2 recieves P1's request, it does not reply because it has a lower timestamp

Distributed Algorithm (cont)



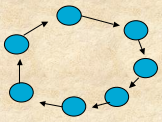
- P1 notes that P2's request has a lower timestamp and so replies immediately
- On receiving a reply from P1 and P3, P2 now has the token!

Evaluation

- Obtaining the token takes 2(n-1) messages.
  - *n-1 to multicast the request*
  - *n-1 replies*
- If hardware support for multicast is available, the obtaining the token requires n messages
- It is fully distributed, but…
  - *failure of **any** node makes progress impossible*
  - *all processes receive and process requests*
    - no performance gain over central algorithm
- Very expensive algorithm!

Ring Based Algorithm

- Arrange *n* processes in a **logical** ring
- exclusion is conferred by obtaining a token in the form of a message passed from process to process in one direction around the ring



- Each process must know the identity of the next process in the ring
- On receiving token, if a process does not need the token, it passes it on immediately

Ring Based Algorithm

- A process that requires the token waits until the token is passed to it, and then retains it.
- To exit the critical section, the process simply passes the token on to its neighbour

Evaluation

- Can take from 1 to n-1 messages to obtain the token
- Token is passed around ring even when nobody needs it!
- No progress possible if process fails

Elections

- If coordinator fails we must choose a new coordinator
- Choice must be unique, even if multiple elections are initiated concurrently

The Bully Algorithm

- Each process is associated with a unique identifier
- Select surviving member with the highest identifier to function as the coordinator
- Members of group know the identity and address of all others
- assumes reliable communication and arbitrary process failure

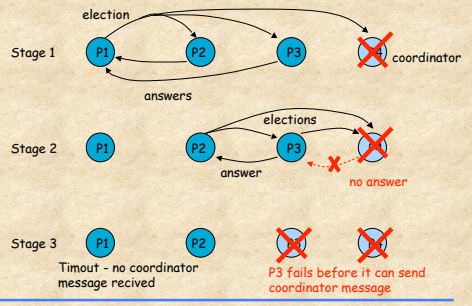
The Bully Algorithm

- Uses three types of message
  - *election messages are sent to announce an election*
  - *answer messages are sent in response to election messages*
  - *coordinator messages are sent to announce the identity of the new coordinator*
- To perform an election…
  - *A process sends an election message to all processes with a higher identifier*
  - *The process awaits answer messages in response*
  - *If none arrive within a certain time, the process considers itself to be the coordinator*
    - sends coordinator messages to all processes with lower identifiers
  - *If answer message does arrive the process waits for a further period for a coordinator message to arrive*
    - If none arrives, it begins a new election
    - If it does arrive, process records identity of new coordinator

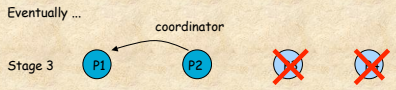
The Bully Algorithm

- If process receives an election message
  - *send back an answer message*
  - *begin a new election, unless it has already begun one*
- If a process fails, and restarts
  - *it begins an election*
  - *will take control if it has the highest process identifier (it's a bully!)*

The Bully Algorithm



The Bully Algorithm



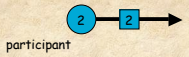
- P2 is elected coordinator after both P4 and P3 fail

Evaluation

- Best case, n-2 messages
- Worst case, $O(n^2)$ messages

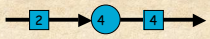
Ring Based Election Algorithm

- Elect the process with the highest identifier as coordinator
- Assume that all processes remain functional and reachable
- Each process knows only about the next node in the chain
- Initially, all nodes are marked as non-participant in an election
- To begin an election
  - *Mark self as participant*
  - *Place identifier in election message*
  - *Send message to next process in chain*

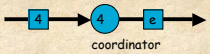


Ring Based Election Algorithm

- If election message is received…
  - *Compare identifiers*
  - *Replace message identifier with own identifier if message identifier is smaller and if not already a participant in the election*



- *forward message and mark self as participant*
- *If identifier is same as message identifier*
  - mark self as coordinator
  - mark self as non-participant
  - send elected message to neighbour



Ring Based Election Algorithm

- If elected message is received at process other than coordinator
  - *mark self as non-participant*
  - *forward message to neighbour*
- If process receives an election message and is already a participant
  - *forward message to neighbour*
- Worst case
  - *anticlockwise neighbour has highest identifier*
  - *3n-1 messages*