Advanced Network Security #3

Mutual Exclusion
Leader election on a ring with Lelann's protocol

But: no FIFO links

node 4 does not receive its own value 10 =) does not know it should be leader
Peterson's leader election: what happens to nodes that do not become a leader?
Peterson leader election algorithm also works on unidirectional rings!

How can that be: nodes send messages clockwise & counterclockwise

1. you can do this if you know $N$ — the number of nodes
   - you forward counterclockwise
   - nodes in the clockwise direction with a step counter

emulation!
Can you design a leader election protocol for general graphs?
Actually, this works too: if protocol is synchronous

\[
V' = h[V_i] \text{ (initial value)} \\
V = \{3\}
\]

\[
\text{while } V \neq V' \rightarrow \\
\text{do } V' = V \rightarrow \\
\text{send } V \text{ to all neighbours} \\
\text{receive } V_i \text{ from all neighbours } j \\
\text{send } V' \rightarrow V_j \rightarrow \\
\text{odd } j \\
\text{leader } \leftarrow V_i = \min V
\]

why - at round \( i \), you receive all values at distance \( i \):

- no new value at round \( i \) \( \Rightarrow \) no node at distance \( i \)

- node at distance \( j \) \( \Rightarrow \) node at distance \( j' \neq j \)
Idea: create a spanning tree with every node as a root forest.
Mutual exclusion

while true
  do
  enter()
  /* critical resource */
  exit() — leave the swimming pool
  /* non critical / remainder section */
properties required

- Mutual exclusion: at most one node in C.S.

- Progress: if at least one node enters, and the C.S. is empty, some node will get into C.S.

- No starvation: if a node enters, and if all nodes that enter the C.S. also release it, it will eventually get access

assumptions

- A fair scheduler
Mutual exclusion using logical clocks

Nodes: keep request queue, initially \([-1, 0]\)

All messages carry a timestamp \(T(m)\) that equals the logical clock value of the send event.

- To request the resource, you send the request to all nodes.
- Nodes that receive a request put it in the request queue. They send an ack back.
- To release the resource, a node sends a release message.
- When you receive this, you remove the request from the queue.
Mutual exclusion with shared variables

⇒ atomic read/write variables

With read-modify-write

flag:

enter() if flag = 0 then flag = 1 and enter
exit() flag = 0
How not to solve mutual exclusion...

Two shared variables  flag_o : written by po, read by p_i
flag_i : written by p_i, read by po

Protocol

\[
\begin{align*}
\text{flag}_i & \leftarrow \text{false} \\
\text{while} & \text{ true} \\
\text{while} (\text{flag}_{i-1} = \text{true}) & \text{ do } A \text{ wait } x \text{ } y \text{ enter } () \\
\text{flag}_i & \leftarrow \text{true} \\
/* \text{ critical section } */ \\
\text{flag}_i & \leftarrow \text{false} \text{ } ] \text{ exit } ()
\end{align*}
\]
Lamport bakery algorithm

node $i$:
- $C[i].num$ : "ticket" $\rightarrow$ unbounded
- $C[i].choosing$ : boolean

idea
- you ask everybody waiting for their ticket and set your ticket to the maximum + 1
- you have to wait for people also computing their ticket!
- lowest ticket gets access.
protocol

\[ (num[i], i) \]

\[ num[i] \leftarrow 0 \]
choosing \[ i \] \leftarrow false

while true
\[
\begin{align*}
\text{do} & \quad \text{true} \\
\text{while} & \quad (num[j] > 0 \\
\text{do} & \quad \text{wait} \* / \\
\text{for} & \quad j \neq i \\
\text{do} & \quad \text{while} \\
\text{while} & \quad (num[j] > 0 \\
\text{do} & \quad \text{wait} \* / \\
\text{critical section} \* /
\end{align*}
\]

\[ num[i] \leftarrow 0 \]

exit

these are \( n \)-independent reads
**Lemma 1**  
If \( i \) is in the critical section and there is \( k \) s.t. \( \text{num}(k) \neq 0 \), then \((\text{num}(k), k) > (\text{num}(i), i)\)

**Proof**  
If \( i \) enters, it first waited until choosing \( k = \text{false} \) and then it waited until either \( \text{num}(k) = 0 \) or \( (\text{num}(i), i) < (\text{num}(k), k) \).

So what if \( \text{num}(k) = 0 \)

**Lemma 2**  
For all \( \text{num}(i) \geq 0 \)

**Lemma 3**  
If \( i \) is in the c.s. then \( \text{num}(i) > 0 \)

**Theorem**  
The bakery protocol satisfies mutual exclusion

**Proof**  
Contradiction: If both \( i \) and \( j \) are in the c.s. then apply Lemma 2.
progress