CS5320 Theory Assignment - 2

Vishal Vijay Devadiga (CS21BTECH11061)

Question 1

The bully algorithm operates effectively on a completely connected graph, assuming:

1. Communication links are fault-free,
2. Processes can only fail by stopping, and
3. Failures can be accurately detected using mechanisms like timeouts.

How could you adapt the bully algorithm to function on a connected graph but is not a completed graph. Since the graph is connected, there exists a path between any two vertices of the graph. Please keep the remaining assumptions unchanged which includes the assumption of a synchronous system.

Solution

The bully algorithm is a leader election algorithm that works as follows:

Suppose a process `p` detects that the leader has failed.

1. `p` sends an election message to all processes with higher IDs than itself.
2. If no process responds, `p` becomes the leader.
3. If a process with a higher ID responds, `p` waits to receive a leader message from some process with a higher ID as it has given up its bid to become the leader.
4. If no leader message is received, `p` restarts the election process.

In case of a connected graph that is not a complete graph, there are two ways to adapt the bully algorithm to work on such graphs:

Method 1: Propagate messages through the graph

Since the graph is connected, there exists a path between any two vertices of the graph. We can use this property to adapt the bully algorithm.

When a process `p` receives a message, it forwards the message to all its neighbors except the sender.

1. `p` sends an election message to all processes with higher IDs than itself either directly or through its neighbors.

2. If no process responds, `p` becomes the leader.
3. If a process with a higher ID responds, `p` waits to receive a leader message from some process with a higher ID as it has given up its bid to become the leader.
4. If no leader message is received, `p` restarts the election process.

This method will eventually terminate as the messages will propagate through the graph and reach all processes.

Method 2: Use a spanning tree

We can construct a spanning tree of the graph and use it to propagate messages.

If a process `p` detects that the leader has failed. A spanning tree is constructed with `p` as the root.

1. `p` sends an election message with the process ID to all its children.
2. If a process with a lower ID receives the message, it forwards the message to all its children.
3. If a process `x` with a higher ID receives the message, it sends a reply message to `p` via the spanning tree. Process `x` now becomes the root of the spanning tree and sends an election message to all its children.
4. If no reply message is received, `p` becomes the leader.
5. If a reply message is received, `p` waits to receive a leader message from some process with a higher ID as it has given up its bid to become the leader. It restarts the election process if eventually no leader message is received.

This method will eventually terminate as in each step, the number of processes that can become the root of the spanning tree decreases as the process with a higher ID becomes the root.

Question 2

In the class, we studied leader election for arbitrary networks for synchronous systems. Can you change this algorithm to work for asynchronous systems?

Solution

The leader election algorithm for arbitrary networks in synchronous systems works as follows:

Initially for each process `p`, `p` is a candidate for leader. Let:

- $L(p)$ denote the leader of process `p`
- $N(p)$ denote the set of neighbors of process `p`

The algorithm runs for D rounds where D is the diameter of the network.

1. For round $r = 1, 2, \ldots, D$:
    1. Each process `p` sends a message to all process in $N(p)$ with $L(p)$.

    2. Each process p updates $L(p)$ to the maximum of the set of leaders received from its neighbors.
2. After D rounds, the process with the maximum ID becomes the leader.

To adapt this algorithm to work for asynchronous systems, we can use the following approach:

Initially for each process p, p is a candidate for leader. Let:

- $L(p)$ denote the leader of process p
- $N(p)$ denote the set of neighbors of process p

Since the system is asynchronous, there must be a **acknowledgment mechanism** to ensure that messages are delivered.

Each process does D rounds of communication.

For each process p, the algorithm works as follows:

1. For round $r = 1, 2, \ldots, D$:
    1. p sends a message to all process in $N(p)$ with $L(p)$.
    2. p waits for acknowledgment from all processes in $N(p)$. If acknowledgment from previous rounds is received, the acknowledgment is ignored.
    3. If acknowledgment for that round is not received from a process q within a timeout, p resends the message to q for that round.
    4. p waits for all messages from $N(p)$. On receiving a message from a process q for round r, p sends an acknowledgment to q for that round. Receiving the same message multiple times, p keeps on sending the acknowledgment for that round.
    5. p updates $L(p)$ to the maximum of the set of leaders received from its neighbors on receiving all messages from $N(p)$.
    6. p moves to the next round.
2. After D rounds, the process with the maximum ID becomes the leader.

This approach ensures that the algorithm works correctly in asynchronous systems by handling message delivery and acknowledgment.