Termination Detection

- Termination detection is an important aspect of distributed computing.
- Simple definition: Check if a distributed computation has Terminated.
- A non-trivial task since no process has complete knowledge of the global state, and global time does not exist.
- Characterized as follows: A distributed computation is globally terminated if every process is locally terminated and there is no message in transit between any processes.
- “Locally terminated” state is a state in which a process has finished its computation and will not restart any action unless it receives a message.
- In the termination detection problem, a particular process (or all of the processes) must infer when the underlying computation has terminated.

Conditions to be satisfied

- Messages used in the underlying computation are called basic messages, and messages used for the purpose of termination detection are called control messages.
- A termination detection (TD) algorithm must ensure the following conditions:
    1. Execution of the TD algorithm cannot indefinitely delay the underlying computation.
    2. The termination detection algorithm must not require addition of new communication channels between processes.

Assumptions About the System

- At any given time, a process can be in only one of the two states: active, where it is doing local computation and idle, where the process has (temporarily) finished the execution of its local computation and will be reactivated only on the receipt of a message from another process.
- An active process can become idle at any time.
- An idle process can become active only on the receipt of a message from another process.
- Only active processes can send messages.
- A message can be received by a process when the process is in either of the two states, i.e., active or idle. On the receipt of a message, an idle process becomes active.
- The sending of a message and the receipt of a message occur as atomic actions.

Termination detection by Weight Throwing

System Model

- A process called *controlling agent* monitors the computation.
- A communication channel exists between each of the processes and the controlling agent and also between every pair of processes.
  Initially, all processes are in the idle state.
- The weight at each process is zero and the weight at the controlling agent is 1.
- The computation starts when the controlling agent sends a basic message to one of the processes.
- A non-zero weight W ($0<W\leq1$) is assigned to each process in the active state and to each message in transit in the following manner:

Termination detection by Weight Throwing

Basic Idea

- When a process sends a message, it sends a part of its weight in the message.
- When a processreceives a message, it adds the weight received in the message to it's weight.
- Thus, the sum of weights on all the processes and on all the messages in transit is always 1.
- When a process becomes passive, it sends its weight to the controlling agent in a control message, which the controlling agent adds to its weight.
- The controlling agent concludes termination if its weight becomes 1.

Notations

- The weight on the controlling agent and a process is in general represented by W.
- B(DW) - a basic message B sent as a part of the computation, where DW is the weight assigned to it.
- C(DW) - a control message C sent from a process to the controlling agent where DW is the weight assigned to it.

Algorithm

The algorithm is defined by the following four rules:

- **Rule 1:** The controlling agent or an active process may send a basic message to one of the processes, say P, by splitting its weight W into W1 and W2 such that W1+W2=W, W1>0 and W2>0. It then assigns its weight W:=W1 and sends a basic message B(DW:=W2) to P.
- **Rule 2:** On the receipt of the message B(DW), process P adds DW to its weight W (W:=W+DW). If the receiving process is in the idle state, it becomes active.
- **Rule 3:** A process switches from the active state to the idle state at any time by sending a control message C(DW:=W) to the controlling agent and making its weight W:=0.
- **Rule 4:** On the receipt of a message C(DW), the controlling agent adds DW to its weight (W:=W+DW). If W=1, then it concludes that the computation has terminated.

Correctness of Algorithm

Notations

- A: set of weights on all active processes
- B: set of weights on all basic messages in transit
- C: set of weights on all control messages in transit
- W_c: weight on the controlling agent.
- Two invariants I_1 and I_2 are defined for the algorithm:
- I_1: $W_c + \sum_{W \in (A \cup B \cup C)} W = 1$
- I_2: $\forall\ W \in (A \cup B \cup C),\ W > 0$

Correctness of Algorithm

- Invariant I_1 states that sum of weights at the controlling process, at all active processes, on all basic messages in transit, and on all control messages in transit is always equal to 1.
- Invariant I_2 states that weight at each active process, on each basic message in transit, and on each control message in transit is non-zero.
- Hence,
  $W_c = 1$
  $\Rightarrow \sum_{W \in (A \cup B \cup C)} W = 0$ (by I_1)
  $\Rightarrow (A \cup B \cup C) = \phi$ (by I_2)
  $\Rightarrow (A \cup B) = \phi$.
- $(A \cup B) = \phi$ implies the computation has terminated. Therefore, the algorithm never detects a false termination.
  Further,
  $(A \cup B) = \phi$
  $\Rightarrow W_c + \sum_{W \in C} W = 1$ (by I_1)
- Since the message delay is finite, after the computation has terminated, eventually $W_c = 1$.
- Thus, the algorithm detects a termination in finite time.