RAMAIAH
Institute of Technology

M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

Course Name: Distributed Systems
Course Code: CSE20
Credits: 3:0:0:1

Term: September – December 2020

Faculty:
Sini Anna Alex

System Model

- 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 using distributed snapshots

The algorithm is defined by the following four rules.

- (R1): When process i is active, it may send a basic message to process j at any time by doing
  send a $B(x)$ *to* j.
- (R2): Upon receiving a $B(x')$, process i does
  let $x{:=}x'{+}1$;
  if (i is idle) $\longrightarrow$ go active.
- (R3): When process i goes idle, it does
  let $x{:=}x{+}1$;
  let $k{:=}i$;
  send message $R(x, k)$ to all other processes;
  take a local snapshot for the request by $R(x, k)$.
- (R4): Upon receiving message $R(x', k')$, process i does
  $[((x', k') > (x,k)) \wedge$ (i is idle)$\longrightarrow$ let $(x,k){:=}(x', k')$;
  take a local snapshot for the request by $R(x', k')$;
  $\square$
  $((x', k') \leq (x,k)) \wedge (i \text{ is idle}) \longrightarrow$ do nothing;
  $\square$
  (i is active)$\longrightarrow$ let $x{:=}\max(x', x)$].
- The last process to terminate will have the largest clock value. Therefore, every process will take a snapshot for it, however, it will not take a snapshot for any other process.

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.

Thank you