Distributed Computing Project Problem Statement

The chosen option is O**ption 1** where we will implement two algorithms for **Termination Detection.**

The algorithms which we have planned to implement are as follows:

Algorithm 1 : A More Efficient Message-Optimal Algorithm for Distributed Termination Detection.
Authors: Ten-Hwang, Lai Yu-Chee Tseng, Xuefeng Dong.
Published in: Parallel Processing Symposium, 1992. Proceedings., Sixth International

The paper has proposed a Termination Detection algorithm that performs better than Spanning Tree(Topor's) and Message-optimal termination detection.

Idea :

- Let S = {P1, P2, P3,....Pn} be the process set of the underlying system which are organized as a spanning tree T, that can be done at compile time. Without loss of generality, let P1 be the root of T.
- Two types of control messages are used in this algorithm i.e **START messages and FINISHED(k) messages**, where k is an integer.
- The START messages are used to "start" the execution of the algorithm. The root starts the algorithm by sending a START message to every other process along the edges of T. The word start is used in the sense of starting to use control messages.
- Before the algorithm starts to operate each process keeps a count of the number of messages it sends i.e. only simple, passive bookkeeping is being done.
- Until the root issues a START message, no control message has ever been sent between processes. The reason for not starting earlier is simply for message efficiency.
- The FINISHED(k) message is used in the message-counting scheme to inform a process that k messages sent by it have been finished.
- The algorithm checks if S is terminated by checking if P1 is "free"(all messages sent to other processes have been acknowledged) and "idle".
- Each process Pi, 1<=i<=n, maintains four variables:
  - **in[1...n]:** An integer array, where $in_i[j]$ counts the number of messages received from Pj which either have not been finished (acknowledged), or have been finished but have not yet been so noticed to Pj. Initially, $in_i[j]$ is 1 if Pj is Pi's parent in T, and 0 otherwise. Note that P1 has no parent.
  - **out:** an integer that indicates the number of unfinished (or thought-to-be-unflnished) messages sent by Pi. Initialize out to the number of children Pi has in T.
  - **mode**: a boolean variable indicating whether Pi is in DT (Detecting Termination) or NDT (Not Detecting Termination) mode. Initially, Pi is in NDT mode. Only after receiving a START message, it changes to DT mode.
  - **parent:** a pointer for a loaded(not free) Pi to indicate where the most recent major message came from. Initially, $parent_i$, i!=1, is initialized to Pi's parent in T, and Parent1 is NULL.

Algorithm:

```
A1: (Upon sending a basic message to Pj)
        out_i := out_i+1;

A2: (Upon receiving a basic message from Pj)
        in_i[j]:= in_i[j]+1;
        if (Parent_i = NULL) AND ( i!=1) then Parent_i := j;

A3: (Upon deciding to switch to DT mode) /* for p1 */
        or (Upon receiving a START message) /* for Pi 2<=i<= n * /
        mode_i := DT;
        for each child Pj of Pi do:
                send a START message to Pj;
        end for
        if (Pi is idle) then:
                call respond_minor(i);
                call respond_major(i);
        end if

A4: (Upon receiving a FINISHED(k) from Pj)
        out_i := out_i - k ;
        if (mode_i=DT) AND (Pi is idle) then call respond_major(i);

A5: (Upon turning idle)
        if (mode_i = DT) then
                call respond_minor(i);
                call respond_major(i);
        end if;

Procedure respond_minor(i: integer)
begin
        for each j!= Parent_i with in_i[j]!=0 do
                send a FINISHED (in_i[j]) to Pj;
                in_i[j]:=0;
        end for ;
end;

Procedure respond_major(i: integer)
begin
        if (out_i=0) then
                if (i=1) then report termination
                else
                        send a FINISHED(in_i[Parent_i]) to Parent_i;
                        in_i[Parent_i]:=0;
                        Parent_i := NULL;
                end if;
        end if;
end;
```

Properties :

- It requires M'+2(n-1) control messages in the worst case and 2(n-1) control messages in the best case, no matter how large M' is, where M' is the number of basic messages and n is the no of nodes in the system.
- Message-Optimal Termination Detection algorithm requires M'+2(n-1)+|E| control messages, whether in best case or worst case, where |E| is the number of edges in the system.
- The worst case detection delay is O(n).
- It works for FIFO Channels.

Algorithm 2 : An efficient delay-optimal distributed termination detection algorithm.
Authors: Nihar R. Mahapatra, Shantanu Dutt.
Published in: Journal of Parallel and Distributed Computing, Volume 67 Issue 10, October, 2007 .

Idea :

- They proposed a new DTD algorithm which uses a spanning termination tree for DTD. The termination tree T is static: it is rooted at a root , with adjacent vertices on the tree corresponding to neighboring PEs in the target topology, and is structured so as to optimize a one-to-all broadcast from the root.
- The control messages used are **STOP, ACKNOWLEDGE, RESUME AND TERMINATION.**
- A non-root node reports a STOP message to its parent once it is free(all messages sent to other processes have been acknowledged) and has received STOP messages from all its child nodes, if any.
- When the root also becomes free, it means that all primary computation is complete, and it signals termination by broadcasting a TERMINATION message to all nodes.
- A primary message $M_{i,j}$ originating at node i and destined for a neighbor node j is said to be “owned” by i until an ACKNOWLEDGE is received for that message.
- If node j has not yet reported a STOP to its parent, then we view the computation load associated with $M_{i,j}$ as being part of the existing primary-computation load at j. In this case, recipient node j sends an ACKNOWLEDGE message to sender node i right away.
- However, if node j has already reported a STOP to its parent before receiving $M_{i,j}$, it “resumes,” sending a RESUME message upward in the termination tree. The RESUME message is sent to nullify a STOP message previously transmitted along this path from j.
- The RESUME message from j travels upward until it encounters an ancestor node k that has not reported a STOP message to its parent (either because it is not free or because it has not received STOP messages from all its children in the termination tree).
- The ancestor node k receiving the last RESUME message then sends an ACKNOWLEDGE for the message $M_{i,j}$ down the termination tree to node j from where it is passed onto the neighboring sender node i.
- On receiving the ACKNOWLEDGE message, i “relinquishes” ownership of $M_{i,j}$ originally sent to j and can report a STOP whenever it becomes free and has received STOPs from all its child nodes.
- When message passing is between arbitrary nodes, the ACKNOWLEDGE message from ancestor k is directly sent to sender node i.

Algorithm:

Algorithm STATIC_TREE_DTD(i)
/* Detects termination of a parallel primary computation on P PEs */
Begin

PE i, $0 \leq i < P$, executes the following steps:

1. Initialization: $idle := 0$; $free := 0$; $inactive := 0$; $\forall$ children j of i, $child\_inactive[j] := 0$; $num\_unack\_msgs := 0$; $terminated = 0$.
/* Here $idle = 1$ (0) $\Rightarrow$ PE i idle (busy); $free = 1$ (0) $\Rightarrow$ PE i free (loaded); $inactive = 1$ (0) $\Rightarrow$ PE i inactive (active); $child\_inactive[j]$ = (# STOP messages) − (# RESUME messages) received from child j of i; num_unack_msgs = # unacknowledged primary messages sent out by i; $terminated = 1$ (0) $\Rightarrow$ PE i has (has not) detected termination. The terms parent and child are used in reference to $\mathcal{T}_{\text{opt}}$. */

Repeat

2. **On** (PE i becoming idle) $idle := 1$.
3. **If** ($idle = 1$) **and** ($num\_unack\_msgs = 0$) **then** $free := 1$.
4. **On** (receiving a STOP from PE j) $child\_inactive[j]++$.
5. **If** ($child\_inactive[j] = 1 \ \forall$ children j of i)
and ($free = 1$) **and** ($inactive = 0$) **then begin**
    If ($i = root$) **then begin**
        Send TERMINATION to all child PEs; $terminated := 1$;
    Endif
    Else Send STOP to parent PE;
    $inactive := 1$;
Endif
6. **On** (issuing a primary message $M_{i,j}$) num_unack_msgs++.
/* $M_{i,j}$ denotes a message from PE i to PE j. */
7. **On** (receiving a primary message $M_{j,i}$) **begin**
    $idle := 0$; $free := 0$;
    If ($inactive = 1$) **then begin**
        Send RESUME$_{j,i}$ to parent PE; $inactive := 0$;
    Endif
    Else Send ACKNOWLEDGE$_{j,i}$ to PE j;
Endon
8. **On** (receiving RESUME$_{j,k}$ from PE l) **begin**
    $child\_inactive[l]--$;
    If ($inactive = 1$) **then begin**
        Send RESUME$_{j,k}$ to parent PE; $inactive := 0$;
    EndIf
    Else Send ACKNOWLEDGE$_{j,k}$ to the child PE on the path to PE k;
Endon
9. **On** (receiving ACKNOWLEDGE$_{j,k}$)
If ($i = j$) **then** num_unack_msgs--;
Else if ($i = k$) **then** Send ACKNOWLEDGE$_{j,k}$ to PE j;
Else Send ACKNOWLEDGE$_{j,k}$ to the child PE on the path to PE k.
10. **On** (receiving TERMINATION) **begin**
    Send TERMINATION to all child PEs; $terminated := 1$;
Endon

Until ($terminated = 1$)

End /* Algorithm STATIC_TREE_DTD */

Properties :

- Message complexity is O(MD+n) where M is the number of basic messages, D is the diameter and n is the no of nodes in the system.
- The best case detection delay is Θ(1) and worse case detection delay is O(D).
- It works for FIFO Channels.

Group Members Information:
Member 1 : Jonathan Marbaniang (CS16MTECH11006)
Member 2 : Mrinal Aich (CS16MTECH11009)