Experiment No. 6-A

Aim– Implement Bully algorithm for leader election.

The Bully algorithm is named "bully" because of its characteristic behavior of higher-priority nodes "bullying" lower-priority nodes during the leader election process. The term "bullying" here refers to the higher-priority nodes asserting their dominance over lower-priority nodes in the election process.

To choose the co ordinator , use election algorithm

Every process is eligible to become a co ordinator

Any process can initiate election algorithm at any point

Every process has a unique id

Only a bigger process can become a coordinator i.e process with higher ids only

The Bully algorithm is a leader election algorithm used in distributed systems to elect a coordinator or leader node among a set of nodes. It allows the nodes to determine the node with the highest priority and elect it as the leader. The algorithm works as follows:

1. Node Hierarchy: Each node in the system is assigned a unique identifier or priority, typically based on its characteristics, such as processing power, capabilities, or ID. The node with the highest priority is considered the leader.
2. Election Initiation: The Bully algorithm assumes that a node initiating the election process has detected the failure of the current leader or coordinator. The initiating node starts the election by sending an election message to all nodes with higher priority than itself.
3. Response from Higher-Priority Nodes: Upon receiving the election message, a higher-priority node that is not aware of a higher-priority node responds to the initiating node with an "OK" message. This response indicates that it is still active and eligible for the leader position.
4. Absence of Higher-Priority Response: If a higher-priority node does not respond within a certain timeout period, the initiating node assumes that the non-responsive node has failed and continues with the election process.
5. Takeover Process: The initiating node, considering itself as the candidate to become the leader, broadcasts a coordinator message to all other nodes, indicating its intention to become the new leader.
6. Acknowledgment: Nodes receiving the coordinator message check if they have a higher priority than the initiating node. If a node finds that its priority is higher, it responds to the coordinator message with an "I'm the Boss" message.
7. New Leader Determination: The initiating node waits for a response from higher-priority nodes. If it receives an "I'm the Boss" message from a higher-priority node, it accepts that node as the leader and steps down from its candidate position.
8. Broadcast of New Leader: If the initiating node does not receive any response from higher-priority nodes, it declares itself as the new leader and broadcasts a new leader message to all nodes, informing them of the new leader's identity.
9. Node Activation: Nodes that receive the new leader message update their knowledge of the leader's identity and activate the necessary procedures to communicate with the new leader.

The Bully algorithm ensures that the highest-priority node in the system becomes the leader through a series of message exchanges and comparisons of priorities. It provides fault tolerance by allowing the system to elect a new leader in the event of leader failure.

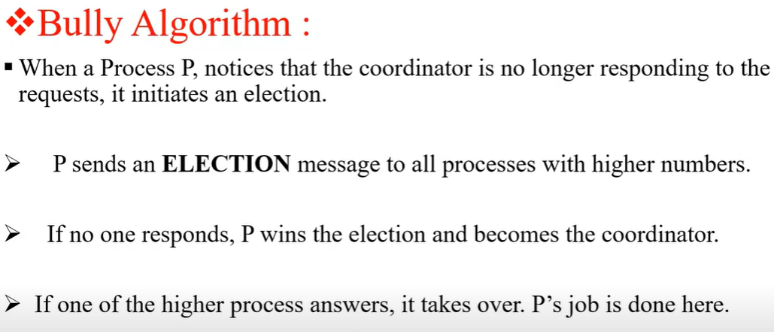
It's important to note that the Bully algorithm assumes a reliable and synchronous communication environment, where message delivery is guaranteed, and node failures can be detected accurately. The algorithm may not be suitable for systems with asynchronous communication or nodes with dynamic behavior.

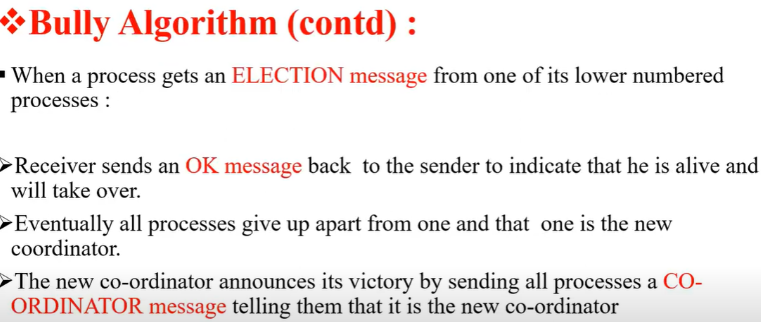
Difference bully and ring

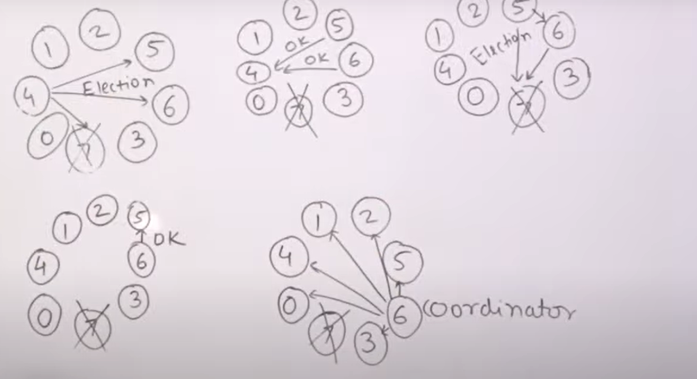
The Bully algorithm and the Ring algorithm are both distributed algorithms used for leader election in a distributed system. However, they differ in terms of their approach and the way they handle leader election.

1. Bully Algorithm:
   * Approach: The Bully algorithm is a centralized approach where the nodes in the system elect the node with the highest priority as the leader.
   * Node Hierarchy: In the Bully algorithm, nodes have a predefined priority or ID, and the node with the highest priority is considered the leader.
   * Election Process: If a node detects that the leader is unavailable, it initiates an election by sending an election message to all nodes with a higher priority. Upon receiving the election message, a higher-priority node takes over the election process, and if no higher-priority nodes respond, the initiating node becomes the leader.
   * Message Complexity: The Bully algorithm has a higher message complexity as each node sends messages to higher-priority nodes during the election process.
   * Scalability: The Bully algorithm may not be suitable for large-scale systems as the message complexity and processing overhead increase with the number of nodes.
2. Ring Algorithm:
   * Approach: The Ring algorithm is a decentralized approach where the nodes form a logical ring and elect a leader based on a token passing mechanism.
   * Node Connectivity: In the Ring algorithm, each node is connected to its adjacent nodes, forming a logical ring structure.
   * Election Process: When a node detects that the leader is unavailable, it passes a token message to its adjacent node. The token circulates around the ring until it reaches the highest ID node, which becomes the leader.
   * Message Complexity: The Ring algorithm has a lower message complexity compared to the Bully algorithm as each node only communicates with its adjacent nodes during the election process.
   * Scalability: The Ring algorithm is more scalable as the message complexity remains constant regardless of the number of nodes in the system. However, it can be slower in large networks due to the token circulation.

In summary, the Bully algorithm is a centralized approach where nodes elect the highest priority node as the leader, while the Ring algorithm is a decentralized approach where nodes form a logical ring and elect a leader based on token passing. The Bully algorithm has higher message complexity and may not be suitable for large-scale systems, while the Ring algorithm has lower message complexity but can be slower in large networks. The choice between the two algorithms depends on the specific requirements and characteristics of the distributed system.







import java.io.\*;

public class Assignment6a

{

int cood,ch,crash; int

prc[];

public void election(int n) throws IOException

{

BufferedReader br=new BufferedReader(new InputStreamReader(System.in)); System.out.println("\nThe Coordinator Has Crashed!");

int flag=1;

while(flag==1)

{

crash=0;

for(int i1=0;i1<n;i1++)

if(prc[i1]==0) crash++;

if(crash==n)

{

System.out.println("\n\*\*\* All Processes Are Crashed \*\*\*"); break;

}

else

{

System.out.println("\nEnter The Intiator"); int

init=Integer.parseInt(br.readLine());

if((init<1)||(init>n)||(prc[init-1]==0))

{

System.out.println("\nInvalid Initiator"); continue;

}

for(int i1=init-1;i1<n;i1++)

System.out.println("Process "+(i1+1)+" Called For Election"); System.out.println("");

for(int i1=init-1;i1<n;i1++)

{

if(prc[i1]==0)

{

System.out.println("Process "+(i1+1)+ " Is Dead");

}

else

System.out.println("Process "+(i1+1)+" Is In");

}

for(int i1=n-1;i1>=0;i1--)

if(prc[i1]==1)

{

cood=(i1+1);

System.out.println("\n\*\*\* New Coordinator Is "+(cood)+" \*\*\*"); flag=0;

break;

}

}

}

}

public void Bully() throws IOException

{

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter The Number Of Processes: ");

int n=Integer.parseInt(br.readLine());

prc=new int[n];

crash=0;

for(int i=0;i<n;i++)

prc[i]=1;

cood=n;

do

{

System.out.println("\n\t1. Crash A Process");

System.out.println("\t2. Recover A Process");

System.out.println("\t3. Display New Cordinator");

System.out.println("\t4. Exit");

ch=Integer.parseInt(br.readLine());

switch(ch)

{

case 1: System.out.println("\nEnter A Process To Crash");

int cp=Integer.parseInt(br.readLine());

if((cp>n)||(cp<1))

{

System.out.println("Invaid Process! Enter A Valid Process");

}

else if((prc[cp-1]==1)&&(cood!=cp))

{

prc[cp-1]=0;

System.out.println("\nProcess "+cp+ " Has Been Crashed");

}

else if((prc[cp-1]==1)&&(cood==cp))

{

prc[cp-1]=0;

election(n);

}

else

{

System.out.println("\nProcess "+cp+" Is Already Crashed");

}

break;

case 2:

System.out.println("\nCrashed Processes Are: \n");

for(int i=0;i<n;i++)

{

if(prc[i]==0)

System.out.println(i+1);

crash++;

}

System.out.println("Enter The Process You Want To Recover");

int rp=Integer.parseInt(br.readLine());

if((rp<1)||(rp>n))

{

System.out.println("\nInvalid Process. Enter A Valid ID");

}

else if((prc[rp-1]==0)&&(rp>cood))

{

prc[rp-1]=1;

System.out.println("\nProcess "+rp+" Has Recovered"); cood=rp;

System.out.println("\nProcess "+rp+ " Is The New Coordinator");

}

else if(crash==n)

{

prc[rp-1]=1;

cood=rp;

System.out.println("\nProcess "+rp+ " Is The New Coordinator"); crash--;

}

else if((prc[rp-1]==0)&&(rp<cood))

{

prc[rp-1]=1;

System.out.println("\nProcess "+rp+" Has Recovered");

}

else

{

System.out.println("\nProcess "+rp+" Is Not A Crashed Process");

}

break;

case 3: System.out.println("\nCurrent Coordinator Is "+cood);

break;

case 4: System.exit(0);

break;

default: System.out.println("\nInvalid Entry!"); break;

}

}

while(ch!=4);

}

public static void main(String args[]) throws IOException

{

Assignment6a ob=new Assignment6a();

ob.Bully();

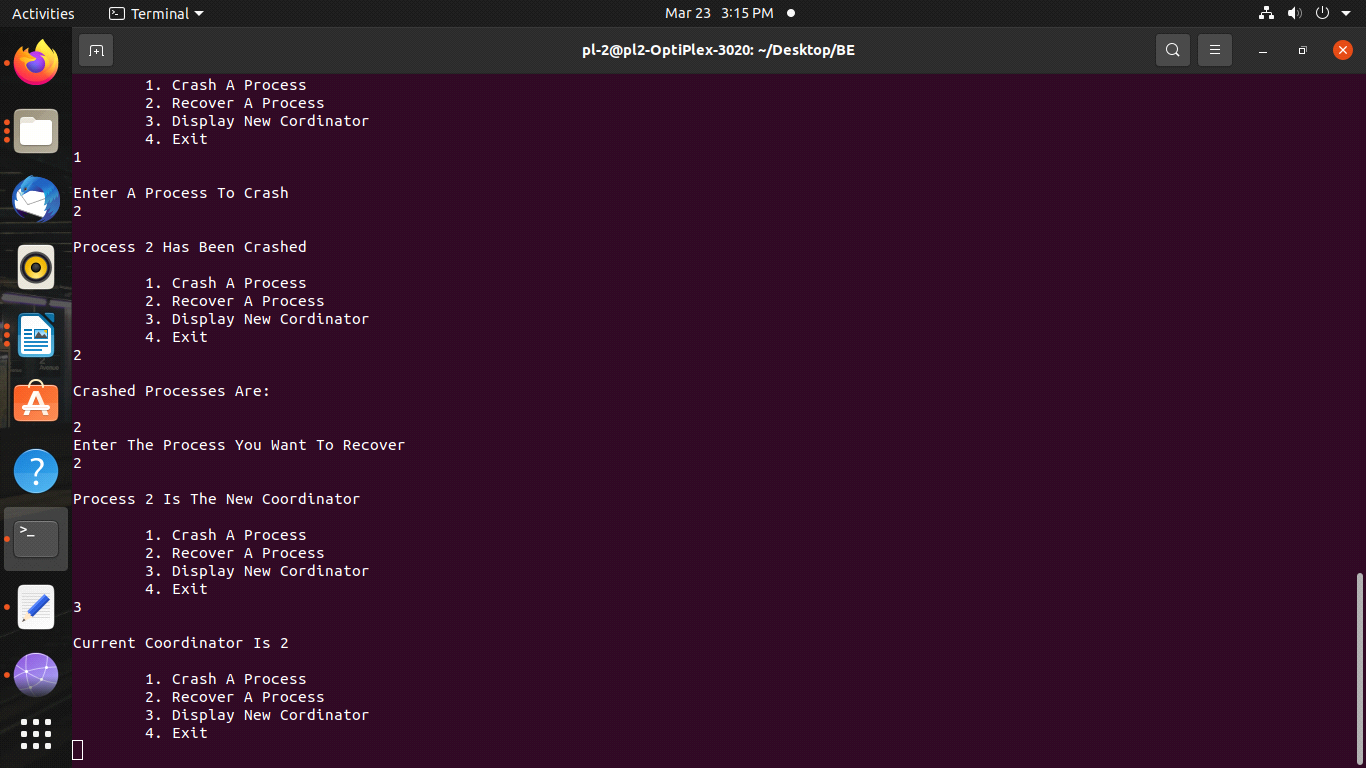
}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[c:\javac](about:blank) BullyAlgo.java

[C:\java](about:blank) BullyAlgo



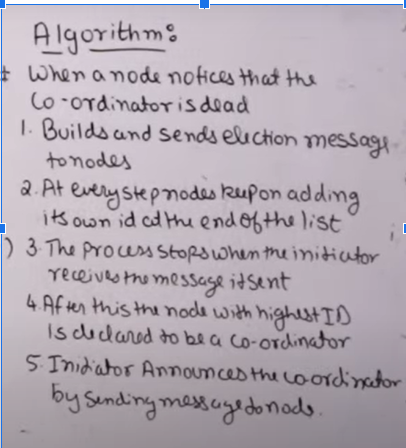
Ring theory

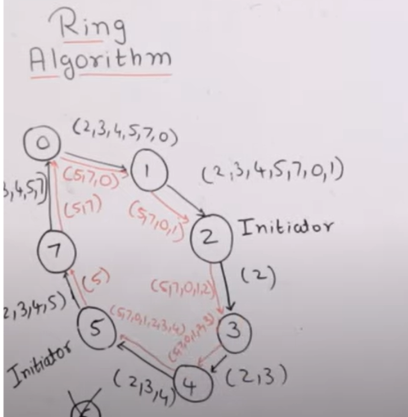
The Ring algorithm is a leader election algorithm used in distributed systems to elect a coordinator or leader node among a set of nodes arranged in a logical ring structure. The algorithm works as follows:

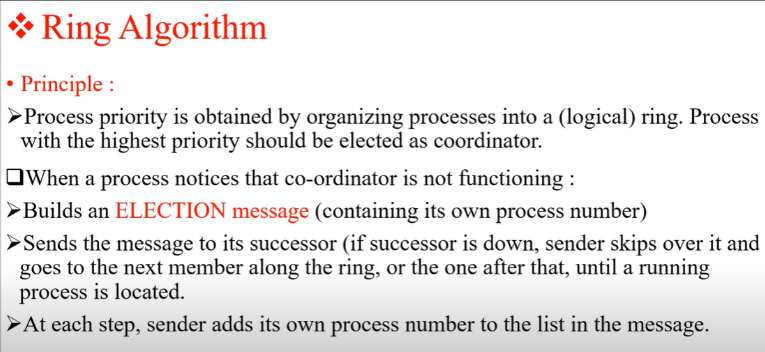
1. Node Initialization: Each node in the system is initialized with a unique identifier or ID. The nodes are arranged in a logical ring structure, where each node has a connection to its adjacent nodes.
2. Election Initiation: When a node detects the failure of the current leader or coordinator, it initiates the election process by sending an election message to its adjacent node(s) in the ring.
3. Token Passing: The election message contains a special token that circulates around the ring. The initiating node starts the circulation by sending the election message along with the token to its adjacent node in a specific direction (clockwise or counterclockwise).
4. Receiving and Forwarding the Token: Upon receiving the token, a node checks its own ID. If its ID is higher than the ID in the token, the node adds its own ID to the token and forwards it to the next node in the ring. If the ID in the token is higher, the node discards the token and does not add its own ID.
5. Token Circulation: The token continues to circulate around the ring, passing through each node. Each node adds its ID to the token if it has a higher ID than the one in the token. This process continues until the token reaches the highest ID node in the ring.
6. Leader Election: The highest ID node in the ring becomes the leader or coordinator. Once the token reaches this node, it stops circulating, and the node declares itself as the new leader.
7. New Leader Announcement: The new leader sends a message to all nodes in the ring, informing them of its identity and the completion of the leader election process.
8. Node Activation: Nodes that receive the new leader message update their knowledge of the leader's identity and activate the necessary procedures to communicate with the new leader.

The Ring algorithm ensures that the node with the highest ID in the ring becomes the leader. The election process is based on the token passing mechanism, where the token circulates around the ring, and each node checks and updates the token with its own ID if necessary.

It's important to note that the Ring algorithm assumes a reliable and synchronous communication environment, where message delivery is guaranteed, and node failures can be detected accurately. Additionally, the algorithm requires that the ring structure remains stable during the leader election process.







ring implementation

import java.util.Scanner;

public class assignment6b {

public static void main(String[] args) {

// TODO Auto-generated method stub

int temp, i, j;

char str[] = new char[10];

Rr proc[] = new Rr[10];

// object initialisation

for (i = 0; i < proc.length; i++)

proc[i] = new Rr();

// scanner used for getting input from console

Scanner in = new Scanner(System.in);

System.out.println("Enter the number of process : ");

int num = in.nextInt();

// getting input from users

for (i = 0; i < num; i++) {

proc[i].index = i;

System.out.println("Enter the id of process : ");

proc[i].id = in.nextInt();

proc[i].state = "active";

proc[i].f = 0;

}

// sorting the processes from on the basis of id

for (i = 0; i < num - 1; i++)

{

for (j = 0; j < num - 1; j++)

{

if (proc[j].id > proc[j + 1].id)

{

temp = proc[j].id;

proc[j].id = proc[j + 1].id;

proc[j + 1].id = temp;

}

}

}

for (i = 0; i < num; i++)

{

System.out.print(" [" + i + "] " + "" + proc[i].id);

}

int init;

int ch;

int temp1;

int temp2;

int ch1;

int arr[] = new int[10];

proc[num - 1].state = "inactive";

System.out.println("\n process" + proc[num - 1].id + "select as co-ordinator");

while (true)

{

System.out.println("\n 1.election 2.quit ");

ch = in.nextInt();

for (i = 0; i < num; i++)

{

proc[i].f = 0;

}

switch (ch)

{

case 1:

System.out.println("\n Enter the Process number who initialsied election : ");

init = in.nextInt();

temp2 = init;

temp1 = init + 1;

i = 0;

while (temp2 != temp1) {

if ("active".equals(proc[temp1].state) && proc[temp1].f == 0) {

System.out.println("\nProcess " + proc[init].id + "send message to " + proc[temp1].id);

proc[temp1].f = 1;

init = temp1;

arr[i] = proc[temp1].id;

i++;

}

if (temp1 == num) {

temp1 = 0;

} else {

temp1++;

}

}

System.out.println("\nProcess" + proc[init].id + "sendmessage to " + proc[temp1].id);

arr[i] = proc[temp1].id;

i++;

int max = -1;

// finding maximum for co-ordinator selection

for (j = 0; j < i; j++) {

if (max < arr[j]) {

max = arr[j];

}

}

// co-ordinator is found then printing on console

System.out.println("\n process " + max + " select as co-ordinator");

for (i = 0; i < num; i++) {

if (proc[i].id == max) {

proc[i].state = "inactive";

}

}

break;

case 2:

System.out.println("Program terminated ...");

return ;

default:

System.out.println("\n invalid response \n");

break;

}

}

}

}

class Rr {

public int index; // to store the index of process

public int id; // to store id/name of process

public int f;

String state; // indiactes whether active or inactive state of node

}

**Run:**

mtt@ubuntu:~/Desktop/Election$ javac Ring.java

mtt@ubuntu:~/Desktop/Election$ java Ring

