****

**Implementation of Bully and Ring Leader Elections in Go**

**INTRODUCTION**

The main purpose of the leader election is to choose a node as a coordinator. It will act as a leader and coordinate activities of the whole system.The election algorithm assumes that every active process in the system has a unique priority number. A leader in any leader election algorithm is usually chosen based on the node which has the largest identifier. Hence, when a coordinator fails, this algorithm elects the active process that has the highest priority number. Then this number is sent to every active process in the distributed system.

**BULLY ALGORITHM FOR LEADER ELECTION**

There are three types of messages in bully algorithm

• An election message is sent to announce an election;

• An answer message is sent in response to an election message and

• A coordinator message is sent to announce the identity of the elected process – the new

‘coordinator’.

* The process that knows it has the highest identifier can elect itself as the coordinator

simply by sending a coordinator message to all processes with lower identifiers. On the other hand, a process with a lower identifier can begin an election by sending an election message to those processes that have a higher identifier and awaiting answer messages in response. If none arrives within time T, the process considers itself the coordinator and sends a coordinator message to all processes with lower identifiers announcing this. Otherwise, the process waits a further period T ` for a coordinator message to arrive from the new coordinator. If none arrives, another election begins.

* If a process pi receives a coordinator message, it sets its variable elected to the identifier of the coordinator contained within it and treats that process as the coordinator. If a process receives an election message, it sends back an answer message and begins anotherelection – unless it has begun one already.
* When a process is started to replace a crashed process, it begins an election. If it has the highest

process identifier, then it will decide that it is the coordinator and announce this to the other

processes. Thus, it will become the coordinator, even though the current coordinator is

functioning. It is for this reason that the algorithm is called the ‘bully’ algorithm.

**RING ALGORITHM FOR LEADER ELECTION**

This algorithm applies to systems organized as a ring (logically or physically). In this algorithm,

we assume that the links between the processes are unidirectional and that every process can

message the process on its right only. The data structure that this algorithm uses is an active list,

a list that has the priority number of all active processes in the system.

In a ring algorithm

* N processes are organized in a logical ring
* Similar to ring in Chord p2p system
* i-th process pi; has a communication channel to 𝑃(𝑖+1) 𝑚𝑜𝑑 𝑁
* All messages are sent clockwise around the ring.

**Algorithm:**

* Any process pi that discovers the old coordinator has failed initiates an "Election” message that contains pi’s own id:attr. This is the initiator of the election.
* When a process pi receives an “Election” message, it compares the attr in the message with its own attr.
  + If the arrived attr is greater, pi forwards the message.
  + If the arrived attr is smaller and pi has not forwarded an election message earlier, it overwrites the message with its own id:attr and forwards it.
  + If the arrived id:attr matches that of pi, then pi's attr must be the greatest and it becomes the new coordinator. This process then sends an "Elected" message to itsneighbor with its id, announcing the election result.

**Benefits and Drawbacks of the Algorithms**

**Both Bully and Ring algorithms are used to elect a coordinator process in distributed systems. Here's a breakdown of their advantages and disadvantages:**

*Bully Algorithm*

Benefits:

* Flexibility: Works in any network topology as processes communicate directly.
* Scalability: Manages well with a larger number of processes compared to Ring.
* Guaranteed leader: Ensures a leader is always elected due to the hierarchical approach.

Drawbacks:

* Complexity: Requires more messages and logic compared to Ring, increasing overhead.
* Single point of failure: Relies on processes knowing each other, so a process failure can disrupt elections.

*Ring Algorithm*

Benefits:

* Simplicity: Easier to implement due to its message-passing nature.
* Fault tolerance: Less susceptible to single process failures as long as the ring remains intact.

Drawbacks:

* Scalability: Performance degrades with a larger number of processes due to message forwarding.
* Limited topology: Restricted to ring-structured networks, limiting its application.
* Potential deadlock: If the highest ID process fails, a new election might not initiate.

In choosing between Bully and Ring, we must consider the following factors:

* Network structure: Bully works for any topology, while Ring needs a ring structure.
* Number of processes: Bully scales better for larger systems.
* Complexity: Ring is simpler to implement.

**Time Complexity**

Bully Algorithm:

* Worst Case: O(n^2)
* Best Case: O(n-2)

Explanation:

* Worst Case: This occurs when the process with the lowest ID initiates the election. It broadcasts an "Election" message to all processes (n-1 messages). Each recipient with a higher ID responds with an "Alive" message (n-2 messages). In the worst case, all processes have higher IDs, leading to a total of 2(n-1) messages.
* Best Case: This occurs when the process with the highest ID (already the leader) initiates the election. It simply broadcasts a "Elected" message to all processes (n-1 messages), resulting in a total of n-1 messages.

Ring Algorithm:

* Time Complexity: O(n)

Explanation:

The Ring algorithm always takes a fixed number of messages (2n-2) regardless of the initiating process or number of processes. This is because the "Election" message travels around the ring once (n-1 messages), and the "Coordinator" message travels back once (n-1 messages).

**Menu Driven Implementation for both Algorithms in GO**

package main

import (

"fmt"

)

const MAX = 20

var pStatus [MAX]int

var n int

var coordinator int

func bully() {

var schoice int

var crash, gid, subcoordinator, flag int

condition := true

for condition {

fmt.Println("---------------------------------------------")

fmt.Println("1.KILL PROCESS\n2.START PROCESS\n3.SHOW PROCESS STATUS\n4.EXIT")

fmt.Println("---------------------------------------------\n")

fmt.Print("Enter your choice: ")

fmt.Scan(&schoice)

switch schoice {

case 1:

fmt.Print("Enter process to crash: ")

fmt.Scan(&crash)

if pStatus[crash] != 0 {

pStatus[crash] = 0

} else {

fmt.Println("Process", crash, "is already dead!\n")

break

}

condition = true

for condition {

fmt.Print("Enter election generator id: ")

fmt.Scan(&gid)

if gid == coordinator || pStatus[gid] == 0 {

fmt.Println("Enter a valid generator id!")

}

condition = gid == coordinator || pStatus[gid] == 0

}

if crash == coordinator {

i := gid + 1

for i <= n {

fmt.Println("Message is sent from", gid, "to", i)

if pStatus[i] != 0 {

subcoordinator = i

fmt.Println("Response is sent from", i, "to", gid)

flag = 1

}

i++

}

if flag == 1 {

coordinator = subcoordinator

} else {

coordinator = gid

}

}

display()

case 2:

var activate int

fmt.Print("Enter Process ID to be activated: ")

fmt.Scan(&activate)

if pStatus[activate] == 0 {

pStatus[activate] = 1

} else {

fmt.Println("Process", activate, "is already alive!")

break

}

if activate == n {

coordinator = n

break

}

flag = 0

i := activate + 1

for i <= n {

fmt.Println("Message is sent from", activate, "to", i)

if pStatus[i] != 0 {

subcoordinator = i

fmt.Println("Response is sent from", i, "to", activate)

flag = 1

}

i++

}

if flag == 1 {

coordinator = subcoordinator

} else {

coordinator = activate

}

display()

case 3:

display()

case 4:

break

default:

fmt.Println("Invalid choice!")

}

condition = schoice != 4

}

}

func ring() {

var tchoice int

var crash, gid, subcoordinator int

condition := true

for condition {

fmt.Println("---------------------------------------------")

fmt.Println("1.KILL PROCESS\n2.START PROCESS\n3.SHOW PROCESS STATUS\n4.EXIT")

fmt.Println("---------------------------------------------\n")

fmt.Print("Enter your choice: ")

fmt.Scan(&tchoice)

switch tchoice {

case 1:

fmt.Print("\nEnter process to crash : ")

fmt.Scan(&crash)

if pStatus[crash] != 0 {

pStatus[crash] = 0

} else {

fmt.Println("Process", crash, "is already dead!")

}

condition = true

for condition {

fmt.Print("Enter election generator id: ")

fmt.Scan(&gid)

if gid == coordinator {

fmt.Println("Please, enter a valid generator id!")

}

condition = gid == coordinator

}

if crash == coordinator {

subcoordinator = 1

for i := 0; i < (n + 1); i++ {

pid := (i + gid) % (n + 1)

if pid != 0 {

if pStatus[pid] != 0 && subcoordinator < pid {

subcoordinator = pid

}

fmt.Println("Election message passed from", pid, ": #Msg", subcoordinator)

}

}

coordinator = subcoordinator

}

display()

case 2:

var activate int

fmt.Print("Enter Process ID to be activated: ")

fmt.Scan(&activate)

if pStatus[activate] == 0 {

pStatus[activate] = 1

} else {

fmt.Println("Process", activate, "is already alive!")

break

}

subcoordinator = activate

for i := 0; i < (n + 1); i++ {

pid := (i + activate) % (n + 1)

if pid != 0 {

if pStatus[pid] != 0 && subcoordinator < pid {

subcoordinator = pid

}

fmt.Println("Election message passed from", pid, ": #Msg", subcoordinator)

}

}

coordinator = subcoordinator

display()

case 3:

display()

case 4:

break

default:

fmt.Println("Invalid choice!")

}

condition = tchoice != 4

}

}

func choice() {

var fchoice int

for {

fmt.Println("---------------------------------------------")

fmt.Println("1.BULLY ALGORITHM\n2.RING ALGORITHM\n3.SHOW PROCESS STATUS\n4.EXIT")

fmt.Println("---------------------------------------------\n")

fmt.Print("Enter your choice: ")

fmt.Scan(&fchoice)

switch fchoice {

case 1:

bully()

case 2:

ring()

case 3:

display()

case 4:

return

default:

fmt.Println("Please, enter valid choice!")

}

}

}

func display() {

fmt.Println("---------------------------------------------")

fmt.Print("PROCESS: ")

for i := 1; i <= n; i++ {

fmt.Printf("%d\t", i)

}

fmt.Print("\nALIVE: ")

for i := 1; i <= n; i++ {

fmt.Printf("%d\t", pStatus[i])

}

fmt.Println("\n---------------------------------------------")

fmt.Println("COORDINATOR IS", coordinator)

}

func main() {

fmt.Print("Enter number of processes: ")

fmt.Scan(&n)

for i := 1; i <= n; i++ {

fmt.Printf("Enter Process %d is alive or not(0/1): ", i)

fmt.Scan(&pStatus[i])

if pStatus[i] != 0 {

coordinator = i

}

}

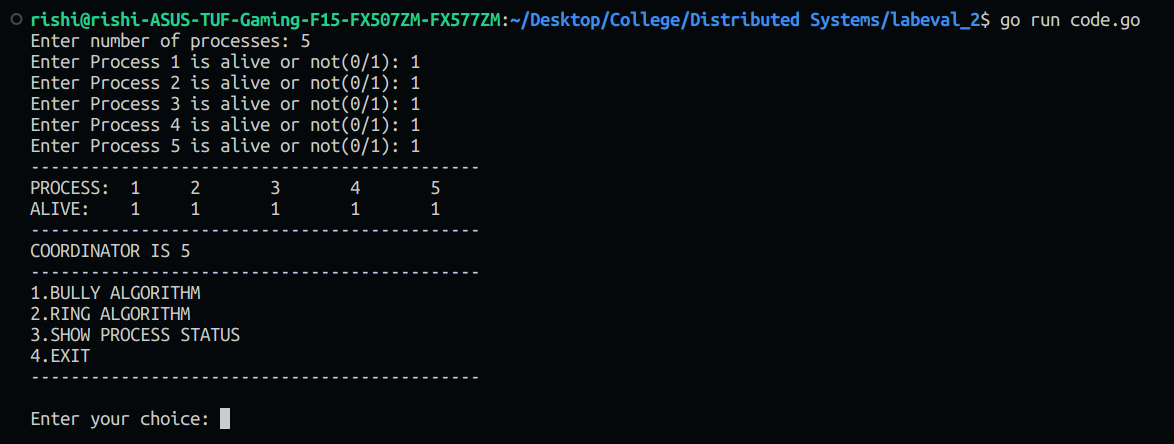
display()

choice()

}

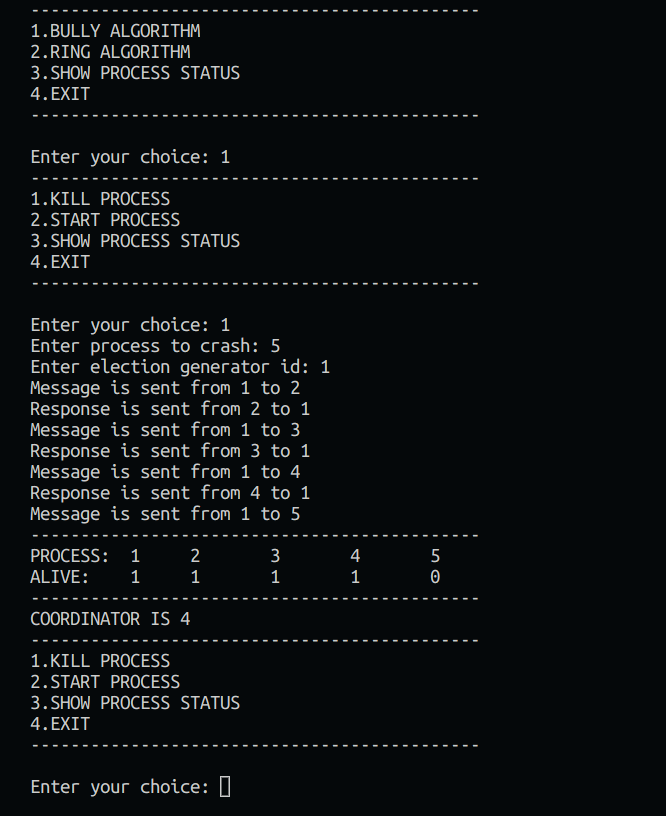
**Output**

Initially we create 5 process and set them all to be alive. The process with the highest priority (in this case process 5) will be selected as the co-ordinator.

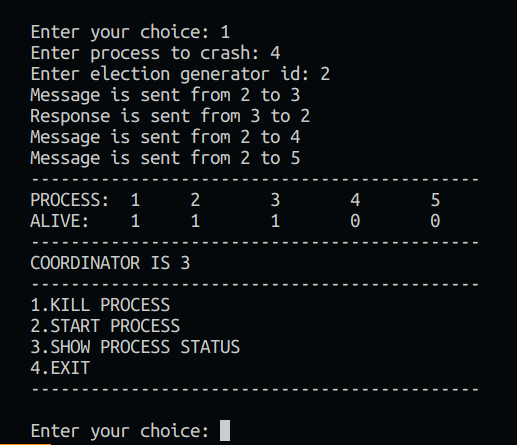


In the next step we select the bully algorithm and select the option to kill

process 5. We then select the option to make process 1 initiate the leader election and we can see the flow of the messages and in the end process 4 which is the current highest priority process becomes the leader

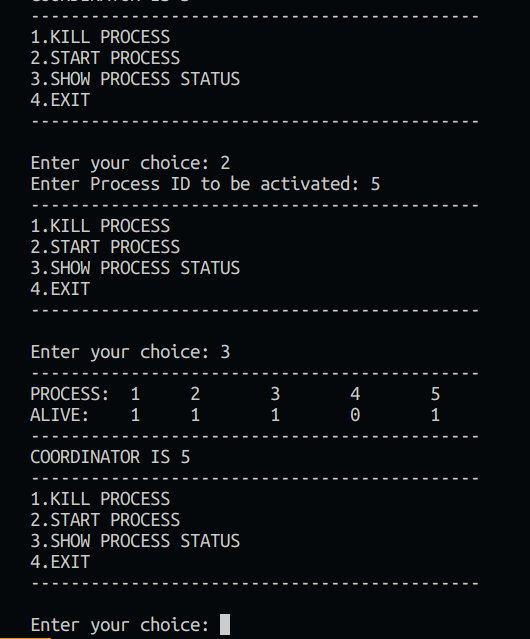


We can see how the message and response have been passed along starting from process 1. Now we can select the option to kill process 4 and initiate an election from process 2. The following is the result produced.

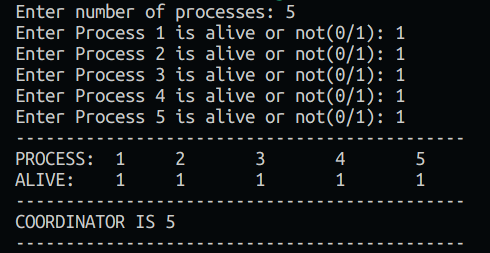


As we can see, process 4 and process 5 are dead which makes process 3 the new leader. The dead processes do not respond to the election message which are initiated by process 2.

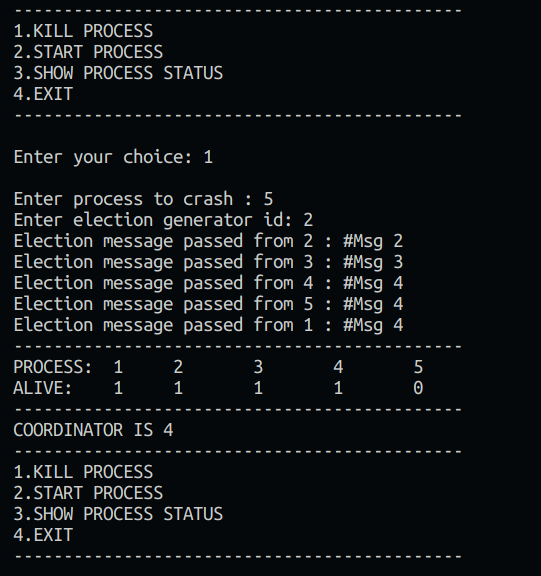
In the next step we are making process 5 alive again which will cause the leader to switch to process 5 as it has the highest priority among the group.



Similary for ring algorithm we initiate 5 processes and set all 5 to be alive. This will cause process 5 to become the leader as it has the highest priority.

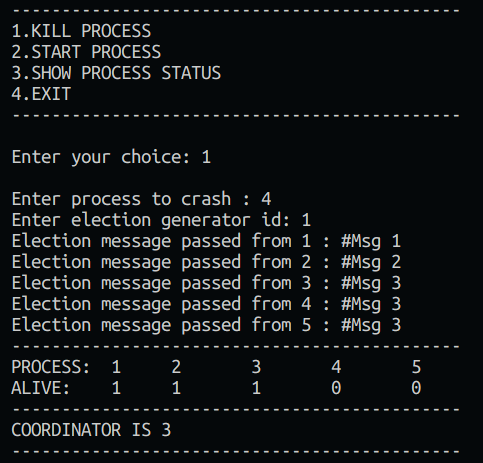


Now we will initiate the ring algorithm by killing process 5 and initiating the leader election from process 2.



In the output above we can see that when process 2 initiates the election it sends the message clockwise through all the processes and process 4 is selected as the new leader.

In the next case we are killing process 4 as well and starting the election from process 1. The output and message flow is as follows



Upon making process 5 alive again it initiates an election and becomes the leader once again. Process 5 sends the election message clockwise until it reaches back.

