**Lab 1: Implementation of Election Algorithm: The Bully Algorithm**

An algorithm for choosing a unique process to play a particular role is called an election algorithm. For example, in a variant of the central-server algorithm for mutual exclusion, the ‘server’ is chosen from among the processes pi, (i = 1, 2,... N) that needs to use the critical section. An election algorithm is needed for this choice. It is essential that all the processes agree on the choice. Afterwards, if the process that plays the role of server wishes to retire then another election is required to choose a replacement.

The election algorithm assumes that every active process in the system has a unique priority number. The process with the highest priority will be chosen as the new coordinator. 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.

We say that a process calls the election if it takes an action that initiates a particular run of the election algorithm. An individual process does not call more than one election at a time, but in principle the N processes could call N concurrent elections. An important requirement is for the choice of the elected process to be unique, even if several processes call for elections concurrently. For example, two processes could decide independently that a coordinator process has failed, and both call elections. Without loss of generality, we require that the elected process be chosen as the one with the largest identifier. The 'identifier’ may be any useful value, as long as the identifiers are unique and totally ordered.

The **bully algorithm** allows processes to crash during an election, although it assumes that message delivery between processes is reliable. This algorithm assumes that the system is synchronous: it uses timeouts to detect a process failure. The bully algorithm also assumes that each process knows which processes have higher identifiers, and that it can communicate with all such processes. There are three types of message in this 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’.

A process begins an election when it notices, through timeouts, that the coordinator has failed. Several processes may discover this concurrently.

**Algorithm**: 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 another election – 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.

**2. Ring Algorithm**

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

• All messages are sent clockwise around the ring.

**Algorithm:**

* Any process that discovers the old coordinator has failed initiates an "Election” message that contains 's own id:attr. This is the initiator of the election.
* When a process ; receives an “Election” message, it compares the attr in the message with its own attr.

• If the arrived attr is greater, , forward the message.

• If the arrived attr is smaller and , 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 ;, then 's attr must be the greatest and it becomes the new coordinator. This process then sends an "Elected" message to its neighbor with its id, announcing the election result.

**Source Code (C++):**

//created by *github.com/yashuv* for Distributed Systems Laboratory Exercise

// first we include the necessary header files

#include <iostream>

#include <cstdlib>

// we define MAX as the maximum number of processes our program can simulate

// we declare array pStatus[MAX] to store the process status; 0 for dead and 1 for alive

// we declare n as the number of processes

// we declare coordinator to store the winner of election

#define MAX 20

int pStatus[MAX], n, coordinator;

using namespace std;

// we declare the necessary functions

void bully();

void ring();

// void ring\_(); // this is also another approach ring implementation, and works well.

void display();

int main()

{

int i, j, fchoice;

cout "Enter the number of processes: ";

cin >> n;

for (i = 1; i <= n; i++)

{

cout << "Enter Process " << i << " is alive or not(0/1): ";

cin >> pStatus[i];

if (pStatus[i])

coordinator = i;

}

display();

do

{

cout << "-------------------------------------------------";

cout << "\n1.BULLY ALGORITHM\n2.RING\n3.DISPLAY\n4.EXIT\n";

cout << "-------------------------------------------------\n\n";

cout << "Enter your choice: ";

cin >> fchoice;

switch (fchoice)

{

case 1:

bully();

break;

case 2:

ring();

// ring\_()

break;

case 3:

display();

break;

case 4:

exit(1);

break;

}

} while (fchoice != 3);

return 0;

}

void display()

{

int i;

// we display the processes, their status and the coordinator

cout << "-------------------------------------------------\n";

cout << "Processes: ";

for (i = 1; i <= n; i++) // PID from 1 to n

cout << i << "\t";

cout << endl

<< "Alive: ";

for (i = 1; i <= n; i++)

cout << pStatus[i] << "\t";

cout << "\n-------------------------------------------------\n";

cout << "COORDINATOR IS " << coordinator << endl;

}

// bully algorithm implementation

void bully()

{

int schoice, crash, activate, i, gid, flag, subcoordinator;

do

{

cout << "-------------------------------------------------";

cout << "\n1.CRASH\n2.ACTIVATE\n3.DISPLAY\n4.EXIT\n";

cout << "-------------------------------------------------\n";

cout << "Enter your choice: ";

cin >> schoice;

switch (schoice)

{

case 1:

// we manually crash the process to see if our implementation

// can elect another coordinator

cout << "Enter process to crash: ";

cin >> crash;

// if the process is alive then set its status to dead

if (pStatus[crash])

pStatus[crash] = 0;

else

cout << "Process " << crash << " is already dead!" << endl;

do

{

// enter another process to initiate the election

cout << "Enter election generator id: ";

cin >> gid;

if (gid == coordinator || pStatus[gid] == 0)

cout << "Please, enter a valid generator id.." << endl;

} while (gid == coordinator || pStatus[gid] == 0);

flag = 0;

// if the coordinator has crashed then we need to find another coordinator

if (crash == coordinator)

{

// the election generator process will send the message to all higher process

for (i = gid + 1; i <= n; i++)

{

cout << "Message is sent from " << gid << " to " << i << endl;

// if the higher process is alive then it will respond

if (pStatus[i])

{

subcoordinator = i;

cout << "Response is sent from " << i << " to " << gid << endl;

flag = 1;

}

}

// the highest responding process is selected as the coordinator

if (flag == 1)

coordinator = subcoordinator;

// else if no higher process are alive then the election generator process

// is selected as coordinator

else

coordinator = gid;

}

display();

break;

case 2:

// enter process to revive

cout << "Enter Process ID to be activated: ";

cin >> activate;

// if the entered process was dead then it is revived

if (!pStatus[activate])

{

pStatus[activate] = 1;

}

else

{

cout << "Process " << activate << " is already alive!" << endl;

break;

}

// if the highest process is activated then it is the coordinator

if (activate == n)

{

coordinator = n;

break;

}

flag = 0;

// else, the activated process sends message to all higher process

for (i = activate + 1; i <= n; i++)

{

cout << "Message is sent from " << activate << " to " << i << endl;

// if higher process is active then it responds

if (pStatus[i])

{

subcoordinator = i;

cout << "Response is sent from " << i << " to " << activate << endl;

flag = 1;

}

}

// the highest responding process is made the coordinator

if (flag == 1)

coordinator = subcoordinator;

// if no higher process respond then the activated process is coordinator

else

coordinator = activate;

display();

break;

case 3:

display();

break;

case 4:

break;

}

} while (schoice != 4);

}

// ring algorithm implementation

void ring()

{

int tchoice, crash, activate, gid, subcoordinator, i;

do

{

cout << "-------------------------------------------------";

cout << "\n1.CRASH\n2.ACTIVATE\n3.DISPLAY\n4.EXIT\n";

cout << "-------------------------------------------------\n\n";

cout << "Enter your choice: ";

cin >> tchoice;

switch (tchoice)

{

case 1:

cout << "\nEnter Process ID to crash : ";

cin >> crash;

if (pStatus[crash])

pStatus[crash] = 0;

else

cout << "Process " << crash << " is already dead!" << endl;

do

{

cout << "Enter election generator id: ";

cin >> gid;

if (gid == coordinator)

cout << "Please, enter a valid generator id.." << endl;

} while (gid == coordinator);

if (crash == coordinator)

{

subcoordinator = 1;

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

{

int pid = (i + gid) % (n + 1);

if (pid != 0) // since process id starts from 1 (to n)

{

if (pStatus[pid] && subcoordinator < pid)

{

subcoordinator = pid;

}

cout << "Election message sent from " << pid << ": #Msg" << subcoordinator << endl;

}

}

coordinator = subcoordinator;

}

display();

break;

case 2:

cout << "Enter Process ID to be activated: ";

cin >> activate;

if (!pStatus[activate])

pStatus[activate] = 1;

else

{

cout << "Process " << activate << " is already alive!" << endl;

break;

}

subcoordinator = activate;

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

{

int pid = (i + activate) % (n + 1);

if (pid != 0)

{

if (pStatus[pid] && subcoordinator < pid)

{

subcoordinator = pid;

}

cout << "Election message passed from " << pid << ": #Msg" << subcoordinator << endl;

}

}

coordinator = subcoordinator;

display();

break;

case 3:

display();

break;

default:

break;

}

} while (tchoice != 4);

}

// ring algorithm implementation (another approach)

// void ring\_()

// {

// int activePID[MAX], PID, k, temp;

// int tchoice, crash, activate, i, generatorid, flag, subordinator;

// do

// {

// cout << "-------------------------------------------------";

// cout << "\n1.CRASH\n2.ACTIVATE\n3.DISPLAY\n4.EXIT\n";

// cout << "-------------------------------------------------\n";

// cout << "Enter your choice: ";

// cin >> tchoice;

// switch (tchoice)

// {

// case 1:

// cout << "\nEnter Process to Crash: ";

// cin >> crash; // process to be crashed id is given

// if (pStatus[crash]) // check whether process is alive or dead

// pStatus[crash] = 0;

// else

// {

// cout << "\nProcess is alreaady dead!!";

// break;

// }

// // election generation

// do

// {

// cout << "\nEnter election generator id:";

// cin >> generatorid;

// if (generatorid == coordinator) // check whether generator is cordinator

// {

// cout << "\nPlease,enter a valid generator id..." << endl;

// }

// } while (generatorid == coordinator);

// flag = 0;

// k = 1;

// if (crash == coordinator) // election is meaningful if crashed process is coordinator

// {

// activePID[k] = generatorid;

// PID = generatorid;

// for (i = (generatorid + 1) % n; i != generatorid; i = (i + 1) % n)

// {

// if (i != 0)

// {

// if (pStatus[i])

// {

// cout << "\nMessage is sent from " << PID << " to " << i;

// activePID[k++] = i;

// PID++;

// }

// }

// }

// subordinator = 1;

// for (i = 1; i < k; i++)

// {

// if (subordinator < activePID[i])

// {

// subordinator = activePID[i];

// }

// }

// coordinator = subordinator;

// for (i = 1; i < k; i++)

// {

// cout << "\n"

// << coordinator << " is coordinator message to " << activePID[i] << endl;

// }

// }

// display();

// break;

// case 2:

// // activate

// cout << "\nEnter Process to Activate: ";

// cin >> activate;

// if (!pStatus[activate]) // activate the process

// pStatus[activate] = 1;

// else

// {

// cout << "\nProcess is already alive!" << endl;

// break;

// }

// if (activate == n) // if highest id process is activated

// {

// coordinator = n; // then automatically it is coordinator

// break;

// }

// temp = activate;

// for (i = (activate + 1) % (n + 1); i != activate; i = (i + 1) % (n + 1))

// {

// if (i != 0)

// {

// cout << "Message is sent from " << temp << " to " << i << endl;

// if (pStatus[i] && i > activate)

// {

// subordinator = i;

// cout << "\nResponse is sent from " << i << " to " << temp << endl;

// flag = 1;

// }

// temp = (temp + 1) % (n + 1);

// if (temp == 0)

// {

// temp++;

// }

// }

// }

// if (flag == 1)

// {

// coordinator = subordinator;

// }

// else

// {

// coordinator = activate;

// }

// display();

// break;

// case 3:

// display();

// break;

// case 4:

// break;

// }

// } while (tchoice != 4);

// }

**PYTHON SOURCE CODE**:

*# we define MAX as the maximum number of processes our program can simulate*

*# we declare pStatus to store the process status; 0 for dead and 1 for alive*

*# we declare n as the number of processes*

*# we declare coordinator to store the winner of election*

MAX = 20

pStatus = [0 for \_ in range(MAX)]

n = 0

coordinator = 0

*# def take\_input():*

*# global coordinator,n*

*# n = int(input("Enter number of processes: "))*

*# for i in range(1, n+1):*

*# print("Enter Process ",i, " is alive or not(0/1): ")*

*# x = int(input())*

*# pStatus[i] = x*

*# if pStatus[i]:*

*# coordinator = i*

def ring():

*global* coordinator, n

condition = True

while condition:

print('---------------------------------------------')

print("1.CRASH\n2.ACTIVATE\n3.DISPLAY\n4.EXIT")

print('---------------------------------------------\n')

print("Enter your choice: ", end='')

tchoice = int(input())

if tchoice == 1:

print("\nEnter process to crash : ", end='')

crash = int(input())

if pStatus[crash]:

pStatus[crash] = 0

else:

print("Process", crash, "is already dead!", end='\n')

condition = True

while condition:

print("Enter election generator id: ", end='')

gid = int(input())

if gid == coordinator:

print("Please, enter a valid generator id!", end='\n')

condition = (gid == coordinator)

if crash == coordinator:

subcoordinator = 1

i = 0

while i < (n+1):

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

if pid != 0: *# since our process starts from 1 (to n)*

if pStatus[pid] and subcoordinator < pid:

subcoordinator = pid

print("Election message passed from", pid, ": #Msg", subcoordinator, end='\n')

i += 1

coordinator = subcoordinator

display()

elif tchoice == 2:

print("Enter Process ID to be activated: ", end='')

activate = int(input())

if not pStatus[activate]:

pStatus[activate] = 1

else:

print("Process", activate, "is already alive!", end='\n')

break

subcoordinator = activate

i = 0

while i < (n+1):

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

if pid != 0: *# since our process starts from 1 (to n)*

if pStatus[pid] and subcoordinator < pid:

subcoordinator = pid

print("Election message passed from", pid,

": #Msg", subcoordinator, end='\n')

i += 1

coordinator = subcoordinator

display()

elif tchoice == 3:

display()

condition = tchoice != 4

def bully():

*global* coordinator

condition = True

while condition:

print('---------------------------------------------')

print("1.CRASH\n2.ACTIVATE\n3.DISPLAY\n4.EXIT")

print('---------------------------------------------\n')

print("Enter your choice: ", end='')

schoice = int(input())

if schoice == 1:

*# we manually crash the process to see if our implementation*

*# can elect another leader*

print("Enter process to crash: ", end='')

crash = int(input())

*# if the process is alive then set its status to dead*

if (pStatus[crash] != 0):

pStatus[crash] = 0

else:

print('Process', crash, ' is already dead!\n')

break

condition = True

while condition:

*# enter another process to initiate the election*

print("Enter election generator id: ", end='')

gid = int(input())

if (gid == coordinator or pStatus[gid] == 0):

print("Enter a valid generator id!")

condition = (gid == coordinator or pStatus[gid] == 0)

flag = 0

*# if the coordinator has crashed then we need to find another leader*

if (crash == coordinator):

*# the election generator process will send the message to all higher process*

i = gid + 1

while i <= n:

print("Message is sent from", gid, " to", i, end='\n')

*# if the higher process is alive then it will respond*

if (pStatus[i] != 0):

subcoordinator = i

print("Response is sent from", i, " to", gid, end='\n')

flag = 1

i += 1

*# the highest responding process is selected as the leader*

if (flag == 1):

coordinator = subcoordinator

*# else if no higher process are alive then the election generator process*

*# is selected as leader*

else:

coordinator = gid

display()

elif schoice == 2:

*# enter process to revive*

print("Enter Process ID to be activated: ", end='')

activate = int(input())

*# if the entered process was dead then it is revived*

if (pStatus[activate] == 0):

pStatus[activate] = 1

else:

print("Process", activate, " is already alive!", end='\n')

break

*# if the highest process is activated then it is the leader*

if (activate == n):

coordinator = n

break

flag = 0

*# else, the activated process sends message to all higher process*

i = activate + 1

while i <= n:

print("Message is sent from", activate, "to", i, end='\n')

*# if higher process is active then it responds*

if (pStatus[i] != 0):

subcoordinator = i

print("Response is sent from", i,

"to", activate, end='\n')

flag = 1

i += 1

*# the highest responding process is made the leader*

if flag == 1:

coordinator = subcoordinator

*# if no higher process respond then the activated process is leader*

else:

coordinator = activate

display()

elif schoice == 3:

display()

elif schoice == 4:

pass

condition = (schoice != 4)

def choice():

""" choice of options """

while True:

print('---------------------------------------------')

print("1.BULLY ALGORITHM\n2.RING ALGORITHM\n3.DISPLAY\n4.EXIT")

print('---------------------------------------------\n')

fchoice = int(input("Enter your choice: "))

if fchoice == 1:

bully()

elif fchoice == 2:

ring()

elif fchoice == 3:

display()

elif fchoice == 4:

exit(0)

else:

print("Please, enter valid choice!")

def display():

""" displays the processes, their status and the coordinator """

*global* coordinator

print('---------------------------------------------')

print("PROCESS:", end=' ')

for i in range(1, n+1):

print(i, end='\t')

print('\nALIVE:', end=' ')

for i in range(1, n+1):

print(pStatus[i], end='\t')

print('\n---------------------------------------------')

print('COORDINATOR IS', coordinator, end='\n')

if \_\_name\_\_ == '\_\_main\_\_':

*# take\_input()*

n = int(input("Enter number of processes: "))

for i in range(1, n+1):

print("Enter Process ", i, " is alive or not(0/1): ")

x = int(input())

pStatus[i] = x

if pStatus[i]:

coordinator = i

display()

choice()

**Analysis**:

Initially, among the five processes, ID:5 is the one with the highest ID, so it is selected as a leader. After that, the process with ID:5 and ID:4 crashes , ID:3 is selected as the leader. After some time, the process with ID:4 activates and calls for election. Since ID:5 is dead, ID:4 is selected as leader. Hence, the process with the highest priority will be chosen as the new leader.

**Conclusion:**

The distributed algorithm, i.e., the election algorithm, was successfully implemented in the IDE and I have deepened my knowledge of distributed processing.