Abstract

Lamport Logical Clock algorithm implementation in C++. The first algorithm is the calculation of Lamport Clocks for all the events in the processor and the Second algorithm is verification of Lamport algorithm. We have successfully implemented the first part of the algorithm, and we were not able to solve the receive event verification on the second part.

Project 1 – Lamport Logical Clock

Submitted By: Ajay Shenoy, Puneet Jain

# Pseudocode

## Algorithm 1: Lamport Algorithm Calculation

1.) Take number of process and events as input.

2.) Initialize Lamport clock to 1.

3.) IF it’s internal or send event.

3.1) If it’s the first event at the process

3.1.1) ASSIGN it’s lamport clock to be 1.

3.2) ELSE if previous value is not known yet

3.2.1) Assign -1 to its lamport value

3.3) ELSE /\* Should be a send event/ internal event\*/

3.3.1) Assign previous event’s lamport clock +1 to current event.

4.) ELSE IF events start with ‘r’ THAN it is RECEIVE event followed by its event number.

4.1) IF it’s the first event at the processor

4.1.1) IF the corresponding send event has not computed Lamport clock

4.1.1.1) ASSIGN -1 to lamport clock

4.1.1) ELSE Calculate the sender lamport clock +1 and ASSIGN it to Lamport clock

4.2)ELSE Calculate sender lamport clock and ASSIGN MAX(sender value, current clock) +1 to it.

5.) ELSE IF the event is NULL then assign 0 to lamport clock

6.) ELSE assign -1 to lamport clock

7.) WHILE all lamport clock value are fixed (i.e no -1)

7.1) Compute lamport clock

8.) Print the results

## Algorithm 2: Lamport Algorithm Verification

1.) Take number of process and lamport clock values as input.

2.) IF lamport\_value == 1 THAN event is internal or send.

2.1) Assign the event as internal event.

2.1.1) IF current\_value is 1 GREATER THAN previous event value than it is an internal event. Else send event.

3.) ELSE IF lamport\_value == 0 THAN it is NULL event.

3.1) Assign the event as NULL.

4.) ELSE it is an receive event.

4.1) Assign the event as receive event.

# Implementation in C++

## Algorithm -1:

//

// Lamport Logical Clock Calculation

//

// Description:

// Given events at the different processes as input the algorithm should

// calculate the logical clock for the events.

//

// Contributers:

// Name: Ajay Hosmar Shenoy, CPSC 474

// Name: Puneet Jain, CPSC 474

//

#include <iostream>

#include <string>

#include <algorithm>

#include<fstream>

using namespace std;

struct events {

string eventType; //Type of the event(i.e Internal/External/Receive)

int clockValue=-1; //Calculated clock value

};

//Function: calculateLamportClock

//Operation: When we have completed calculating Lamport clock for every Send and Internal events. We start calculating Lamport Clock values for Receive events.

//Declaration

void calculateLamportClock(events clock\_temp[10][10], int n, int m, int lamportClock[10][10]);

//Function: calculateSendEventClock

//Operation: Match the correspondent send event and return the Lamport Logical clock.

//Declaration

int calculateSendEventClock(events clock\_temp[10][10], int n, int m, string x);

bool checkAll(events clock\_temp[10][10], int n, int m, int lamportClock[10][10]);

int main()

{

int n=0; //n= number of processes

int m=0; //m=max number of events

int lc\_value=1;

events clock\_temp[10][10];

int lamportClock[10][10]; //Clock values

//Input for all the processes

cout << "Enter number of Processes: ";

cin >> n;

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

{

cout << "Enter number of events per process p" << x+1 << ":";

cin >> m;

for (int y = 0; y < m; y++)

{

cout << "p" << x+1 << ":";

cin >> clock\_temp[x][y].eventType;

}

}

//Computation of the algorithm

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

{

for (int j = 0; j < m; j++)

{

if (clock\_temp[i][j].eventType.length()==1 || clock\_temp[i][j].eventType[0]=='s') //Check if it is Internal or Send Event

{

if (j == 0)//Check if it is the first event at the processor

{

clock\_temp[i][j].clockValue = 1; //Assign LC=1 for all the first event

lamportClock[i][j] = clock\_temp[i][j].clockValue;

}

else if(clock\_temp[i][j-1].clockValue==-1)

{

clock\_temp[i][j].clockValue=-1;

//lamport=-1??

}

else

{

lc\_value = clock\_temp[i][j - 1].clockValue + 1;

clock\_temp[i][j].clockValue = lc\_value; //Save the value for that particular event in temp

lamportClock[i][j] = lc\_value;

lc\_value++;

}

}

else if (clock\_temp[i][j].eventType.length() > 1 && clock\_temp[i][j].eventType[0]=='r') //Check if it is a Receive event

{

if (j == 0) //Check if it is the first event at the processor

{

int k = calculateSendEventClock(clock\_temp, n, m, clock\_temp[i][j].eventType); //Returns the clock value of send event

if (k == -1)

{

clock\_temp[i][j].clockValue = k; // Wait for the corresponding send event

}

else

{

clock\_temp[i][j].clockValue = k + 1;

lamportClock[i][j] = k + 1;

lc\_value++;

}

}

else

{

int send\_event = calculateSendEventClock(clock\_temp, n, m, clock\_temp[i][j].eventType); //Calculate send event LC Value

if (send\_event >0)

{

int k = clock\_temp[i][j - 1].clockValue; //The event before the receive event in that processor

clock\_temp[i][j].clockValue = max(k, send\_event) + 1;

lamportClock[i][j] = clock\_temp[i][j].clockValue;

lc\_value = clock\_temp[i][j].clockValue;

lc\_value++;

}

else

{

//Set the Value to be -1 for now. Will be updated when send events are processed

clock\_temp[i][j].clockValue = -1;

lamportClock[i][j] =-1;

}

}

}

else if (clock\_temp[i][j].eventType == "0" || clock\_temp[i][j].eventType == "NULL" || clock\_temp[i][j].eventType == "null" ) //No value

{

clock\_temp[i][j].clockValue = 0;

lamportClock[i][j] = 0;

}

else

{

lamportClock[i][j]=-1;

clock\_temp[i][j].clockValue=-99;

}

}

}

while(!checkAll(clock\_temp, n, m, lamportClock))

{

calculateLamportClock(clock\_temp, n, m, lamportClock);

}

//Print Lamport Clock Values

cout<<endl<<"Lamport Logical Clock: "<<endl;

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

{

cout << "P" << i+1 << ": ";

for (int j = 0; j < m; j++)

{

if(lamportClock[i][j]!=-1){

cout<< lamportClock[i][j]<<"\t";

}

}

cout<< endl;

}

ofstream outfile("a1output.txt");

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

{

outfile << "P" << i+1 << ":";

for (int j = 0; j < m; j++)

{

if(lamportClock[i][j]!=-1){

outfile<< lamportClock[i][j]<<" ";

}

}

outfile << endl;

}

return 0;

}

//Function Implementation

void calculateLamportClock(events clock\_temp[10][10], int n, int m, int lamportClock[10][10])

{

int send\_event = 0;

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

{

for (int j = 0; j < m; j++)

{

if (clock\_temp[i][j].clockValue == -1)

{

if (clock\_temp[i][j].eventType.length() == 1 || clock\_temp[i][j].eventType[0]=='s') //Checking Internal event

{

if(clock\_temp[i][j-1].clockValue!=-1)

{

clock\_temp[i][j].clockValue = clock\_temp[i][j - 1].clockValue + 1; //Calculating LC for Internal Event

lamportClock[i][j] = clock\_temp[i][j].clockValue;

}

}

else //Checking Receive event

{

send\_event = calculateSendEventClock(clock\_temp, n, m, clock\_temp[i][j].eventType);

if(send\_event!=-1)

{

clock\_temp[i][j].clockValue = max(clock\_temp[i][j - 1].clockValue, send\_event) + 1; //Calculating LC for Receive Event

lamportClock[i][j] = clock\_temp[i][j].clockValue;

}

}

}

}

}

}

int calculateSendEventClock(events clock\_temp[10][10], int n, int m, string x)

{

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

{

for (int j = 0; j < m; j++)

{

if (x[1] == clock\_temp[i][j].eventType[1]) //Checking the index of send values. i.e S1

{

if(x[0]==clock\_temp[i][j].eventType[0])

{

}

else {

if(clock\_temp[i][j].clockValue==-1)

{

return -1; //If corresponding send event is not found

}

else

{

return clock\_temp[i][j].clockValue;

}

}

}

}

}

}

bool checkAll(events clock\_temp[10][10], int n, int m, int lamportClock[10][10])

{

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

{

for (int j = 0; j < m; j++)

{

if (clock\_temp[i][j].clockValue == -1)

{

return false;

}

}

}

return true;

}

## Algorithm-2:

//

// Lamport Logical Clock Verification

//

// Description:

// Given Lamport logical clock input the algorithm should

// find the events associated with those logical clock

//

// Contributers:

// Name: Ajay Hosmar Shenoy, CPSC 474

// Name: Punith Jain, CPSC 474

//

#include <iostream>

#include <string>

#include <algorithm>

using namespace std;

//Function:checkInternalSend

//Operation: Check if operation is Internal

//Declaration

bool checkInternal(int n, int m, int value, int clock\_temp[10][10]);

//Function:checkSend

//Operation: Check if operation is Send

//Declaration

bool checkSend(int n, int m, int value, int clock\_temp[10][10]);

//Function:checkReceive

//Operation: Check if operation is Receive

//Declaration

bool checkReceive(int n, int m, int value, int clock\_temp[10][10], string events[10][10]);

//Algorithm Verify

int main()

{

int n=0;

int m=0;

bool correct=true;

string internal\_events[9] = { "a","b","c","d","e","f","g","h","i" };

string send\_events[9] = { "s1","s2","s3","s4","s5","s6","s7","s8","s9" };

string receive\_events[9] = { "r1","r2","r3","r4","r5","r6","r7","r8","r9" };

string events[10][10];

int lamportClock[10][10]; //Stores the values

//Input for all the processes

cout << "Enter number of Processes: ";

cin >> n;

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

{

cout << "Enter number of events per process p" << x+1 << ":";

cin >> m;

for (int y = 0; y < m; y++)

{

cout << "p" << x+1 << ":";

cin >> lamportClock[x][y];

}

}

int internal\_counter = 0;

int send\_counter = 0;

int receive\_counter = 0;

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

{

for (int j = 0; j < m; j++)

{

if (lamportClock[i][j] == 1) //It should be internal or send event

{

int value = lamportClock[i][j];

if (checkInternal(n, m, value, lamportClock)) //Verify it is internal or send event

{

events[i][j] = internal\_events[internal\_counter];

internal\_counter++;

}

else

{

events[i][j] = send\_events[send\_counter];

send\_counter++;

}

}

else if (lamportClock[i][j] == 0)

{

events[i][j] == "NULL";

}

else

{

int next = lamportClock[i][j + 1];

int value = lamportClock[i][j];

if (j == 0) //First event at the process

{

if (checkReceive(n, m, value, lamportClock,events)) //A receive event

{

events[i][j] = receive\_events[receive\_counter];

receive\_counter++;

}

else if (checkInternal(n,m,value,lamportClock))

{

events[i][j] = internal\_events[internal\_counter];

internal\_counter++;

}

}

if (next==lamportClock[i][j]+1) //when its a receive or send event

{

events[i][j] = send\_events[send\_counter];

send\_counter++;

events[i][j+1] = internal\_events[internal\_counter+1];

}

else

{

correct = false;

//break;

}

}

}

}

cout<<endl<<"Lamport Logical Clock Verification: "<<endl;

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

{

cout << "P" << i+1 << ":";

for (int j = 0; j < m; j++)

{

cout << events[i][j]<<"\t";

}

cout<<endl;

}

if(correct)

{

cout<< "Correct"<<endl;

}

else

{

cout << "Incorrect" << endl;

}

}

//Function Implementations

bool checkInternal(int n, int m, int value, int clock\_temp[10][10])

{

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

{

for (int j = 0; j < m; j++)

{

if (value == clock\_temp[i][j])//This means its an internal because they are concurrent

{

return true;

}

}

}

return false;

}

bool checkSend(int n, int m, int value, int clock\_temp[10][10])

{

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

{

for (int j = 0; j < m; j++)

{

if (value == clock\_temp[i][j])

{

return true;

}

}

}

return false;

}

bool checkReceive(int n, int m, int value, int clock\_temp[10][10], string events[10][10])

{

string target= "s";

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

{

for (int j = 0; j < m; j++)

{

if(events[i][j].find(target)!=string::npos)

{

if (value ==clock\_temp[i][j]) //if this happens its for sure not a receive event

{

return false;

}

}

}

}

return true;

}