

Implementation of Chandy-Lamport algorithm

By

Vikram saha(Roll No: 001611001016)

Kaushik Rajak(Roll No:001611001029)

Under the guidance of

Mr. Utpal Kumar Ray

Project Report submitted for the Bachelor's Degree in
Information Technology, B.E.,
4th year, 2016–2020,
Jadavpur University



Department of Information Technology,
Faculty of Engineering and Technology,
Jadavpur University,
Kolkata, India
2016-2020

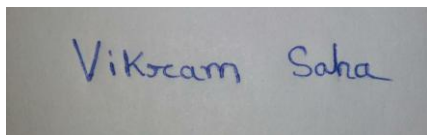
Declaration

We hereby declare that this project report titled “Implementation of Chandy- Lamport algorithm”, contains only the work completed by us as a part of the Bachelor’s of Engineering course, during the year 2019-2020, under the supervision of Mr. [Utpal Kumar Ray](#), Department of Information Technology, Jadavpur University.

All information, materials, and methods that are not original to this work have been properly referenced and cited.

We also declare that no part of this project work has been submitted for the award of any other degree prior to this date.

Signature :

A rectangular box containing a handwritten signature in blue ink that reads "Vikram Saha".A handwritten signature in black ink that reads "Kaushik Rajak".

Date : 12.06.2020

Certificate

This is to certify that the project entitled “Implementation of Chandy-Lamport algorithm”, carried out by Vikram saha(001611001016) and Kaushik Rajak(001611001029) and submitted to the Jadavpur University during the year 2019-2020 for the award of the degree of Bachelor’s of Engineering, is a bonafide record of work done by them under my supervision. The contents of this project, in full or in parts, have not been submitted to any other Institute or University for the award of any other degree.

Signature of the guide :

Date :

Acknowledgment

First and foremost we would like to express our deep and sincere gratitude to our Project guide Mr. Utpal Kumar Ray, Assistant Professor of the Information Technology Department, Jadavpur University for giving us this opportunity to work on an equally challenging and enjoyable project under his wonderful supervision.

Without his continuous support, encouragement, and valuable guidance, the project would not have been as interesting and bounteous to us.

Furthermore, we would also like to thank Mr. Rayan Ray for his extensive help in the beginning of the project.

Thank you.

Contents

1. Introduction	6
2. Pre-Requisites	8
2.1 Open MPI	8
2.2 Multithreading in C	10
3. Problem Description	12
4. Approach	13
4.1 Workflow	14
5. Evaluation Methodology & Results	17
6. Conclusion	20
7. References	21
8. Appendix: System Description	22

1.0 INTRODUCTION

The Chandy–Lamport algorithm is a snapshot algorithm that is used in distributed systems for recording a consistent global state of an asynchronous system. It was developed by and named after Leslie Lamport and K. Mani Chandy.

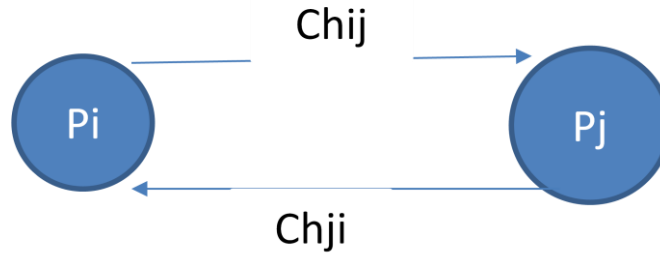
Each distributed system has a number of processes running on a number of different physical servers. These processes communicate with each other via communication channels using text messaging. These processes neither have a shared memory nor a common physical clock, this makes the process of determining the instantaneous global state difficult.

Any process in the distributed system can initiate this global state recording algorithm using a special message called **MARKER**. This marker traverses the distributed system across all communication channels and causes each process to record its own state. In the end, the state of the entire system (Global state) is recorded. This algorithm does not interfere with normal execution of processes.

Assumptions of the algorithm:

- There are finite number of processes in the distributed system and they do not share memory and clocks.
- There are finite number of communication channels and they are unidirectional and FIFO ordered.
- There exists a communication path between any two processes in the system
- On a channel, messages are received in the same order as they are sent.

Algorithm:



1. Initiator process P_0 records its state locally

2. Marker sending rule for process P_i :

-After P_i has recorded its state, for each outgoing channel Ch_{ij} , P_i sends *one marker* message over Ch_{ij} (before P_i sends any other message over Ch_{ij})

3. Marker receiving rule for process P_i :

-Process P_i on receipt of a marker over channel Ch_{ji}

-If (P_i has not yet recorded its state)

--It Records *its process state* now;

--Records the state of Ch_{ji} as *empty set*;

--Starts recording messages arriving over other incoming channels;

-else (P_i has already recorded its state)

-- P_i records the state of Ch_{ji} as the set of all messages it has received over Ch_{ji} since *it* saved its state

Need of taking snapshot or recording global state of the system:

- **Checkpointing:** It helps in creating checkpoint. If somehow application fails, this checkpoint can be used restart.
- **Garbage collection:** It can be used to remove objects that do not have any references.
- It can be used in deadlock detection and termination.
- It is also helpful in other debugging.

2.0 PRE REQUISITES

(Overview of required pre requisites are explained below.)

- OPEN MPI (Version 4.0.0)
- C Language
- Linux Operating System (Fedora or Ubuntu)

Some pre requisites concepts are explained henceforth.

2.1 OPEN MPI

The Open MPI Project is an open source Message Passing Interface implementation that is developed and maintained by a consortium of academic, research, and industry partners. Open MPI is therefore able to combine the expertise, technologies, and resources from all across the High Performance Computing community in order to build the best MPI library available. Open MPI offers advantages for system and software vendors, application developers and computer science researchers.

Here, this technology is basically used to create a distributed environment so that we can make our algorithm and run on it.

To building an MPI program first we need to add MPI header file with

```
#include<mpi.h>
```

After this, the MPI environment must be initialized with:

```
MPI_Init(  
    int* argc,  
    char*** argv)
```

During **MPI_Init()**, all of MPI's global and internal variables are constructed.

After `MPI_Init`, there are two main functions that are called. These two functions are used in almost every single MPI program that you will write.

- `MPI_Comm_size(MPI_Comm communicator, int* size)`
- `MPI_Comm_rank(MPI_Comm communicator, int* rank)`

`MPI_Comm_size()` returns the size of a communicator.

`MPI_Comm_rank()` returns the rank of a process in a communicator. Each process inside of a communicator is assigned an incremental rank starting from zero. The ranks of the processes are primarily used for identification purposes when sending and receiving messages.

The final call in a MPI program is:

- `MPI_Finalize()`

`MPI_Finalize()` is used to clean up the MPI environment. No more MPI calls can be made after this one.

2.2 MULTITHREADING IN C

In a **Linux operating system**, the **C languages** provide the POSIX thread (pthread) standard API(Application program Interface) for all thread related functions. It allows us to create multiple threads for concurrent process flow. It is most effective on multiprocessor or multi-core systems where threads can be implemented on a kernel level for achieving the speed of execution.

In our project we have created two thread for sending and receiving token simultaneously.

Some of the important functions of pthread which are used in our project are discussed below:

1)

Syntax:

```
int pthread_create(pthread_t * thread,  
                  const pthread_attr_t * attr,  
                  void * (*start_routine)(void *),  
                  void *arg);
```

- Pthread_create() function is used to create a new thread.

Parameter:

- **thread:** pointer to an unsigned integer value that returns the thread id of the thread created.
- **attr:** pointer to a structure that is used to define thread attributes like detached state, scheduling policy, stack address, etc. Set to NULL for default thread attributes.
- **start_routine:** pointer to a subroutine that is executed by the thread. The return type and parameter type of the subroutine must be of type void *. The function has a single attribute but if multiple values need to be passed to the function, a struct must be used.
- **arg:** pointer to void that contains the arguments to the function defined in the earlier argument

2)

Syntax:

```
void pthread_exit(void *retval);
```

- Pthread_exit() function is used to terminate a thread

Parameters: This method accepts a mandatory parameter **retval** which is the pointer to an integer that stores the return status of the thread terminated. The scope of this variable must be global so that any thread waiting to join this thread may read the return status.

3)

pthread_join: used to wait for the termination of a thread.

Syntax:

```
int pthread_join(pthread_t th, void **thread_return);
```

Parameter: This method accepts following parameters:

- **th:** thread id of the thread for which the current thread waits.
- **thread_return:** pointer to the location where the exit status of the thread mentioned in th is stored.

3.0 PROBLEM DESCRIPTION

Day by day we are trying to eliminate centralized system by implementing distributed system. To implement distributed system we require many algorithms. These algorithms can create a centralized view of a distributed system.

Network is the main factor by which we can connect all the distributed components of a system. There are many networking algorithm for distributed systems. These algorithms help us in establishing different type of connections between components.

But in implementing distributed system we have to come across many problems. One of such problems is about consistency of the system. One solution for checking consistency of a system is by taking a snapshot of the system. To take a snapshot of a distributed system we can implement Chandy lamport algorithm. Our project is all about this algorithm and to implement this algorithm in a small distributed system. To create a distributed system we have used mpi.

4.0 APPROACH

Before implementing the algorithm we need to make a distributed system environment. Here we use OPEN MPI as the underline technology to connect several computers with Linux operating system.

We have done this project in two phases.

Phase 1:

Installing OPEN-MPI in multiple computer and making a distributed system.

There are several distributed system possible in the real world. Here we choose a money transfer system as a distributed system where each node in the system holding equal amount of money at the beginning. After running the program each node sending an random amount of its money to another random nodes.

To implement the money transfer system, we have used two thread in each node one for sending money another for receiving money. Then with the help of MPI library function we send random amount of money to randomly selected node.

In phase 1 our goal is to check whether the money transfer program working properly or not. After the stopping the program total amount of the system should remain same.

Phase 2:

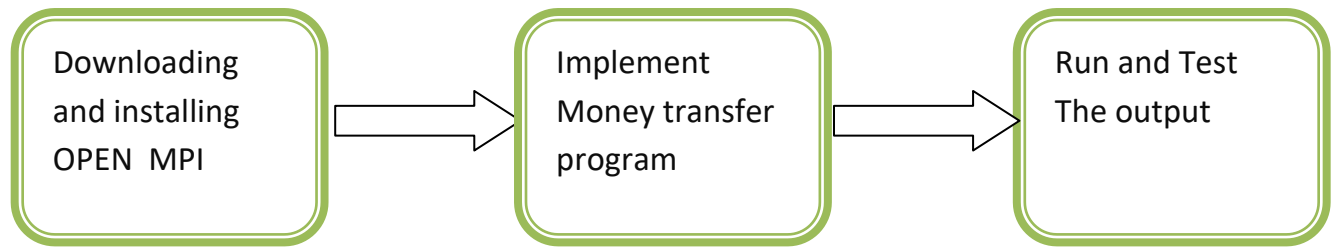
After successfully completing phase 1, we will move to phase 2 where our goals are – implementing the algorithm and print the total amount of the system.

Form implementation of the algorithm we have define the sending rule of algorithm in sending thread and receiving rule of the algorithm in receiving thread.

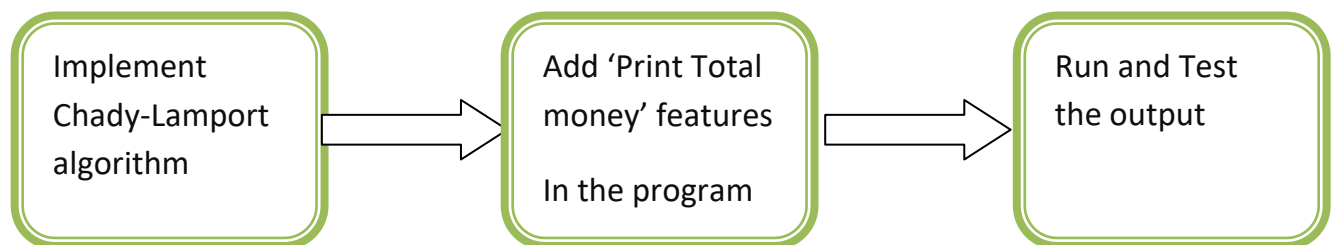
Now pressing '1' in the key board we can see the output in the terminal.

4.1 WORKFLOW

PHASE 1:



PHASE 2:



PHASE-I WORKFLOW ACHIEVED:

- **Download and install OPEN MPI :**

We have download the OPEN MPI version 4.0.0 from the official site “<https://www.open-mpi.org/software/ompi/v4.0/> “ and install in our multiple linux system (fedora).

These are some important command for installation:

- `gunzip -c openmpi-4.0.0.tar.gz | tar xf -`
- `cd openmpi-4.0.0`
- `./configure --prefix=/usr/local`
- `make all install`

- **Implement Money Transfer Program :**

This program is implemented using two thread one is used for sending money to randomly selected node another is for receiving money from any other node.

- **Run and test :**

For testing the output we have run the program for about two and half an hour and capture the output.

For testing we have to check the total money should be same at the end of the program.

PHASE-II WORKFLOW ACHIEVED:

- **Implement Chady-Lamport algorithm :**

After successfully testing the money transfer program we have extended the program further to implement the algorithm.

We have made some changes in the sending and receiving thread.

- Sending marker message with tag '7'
- Make a queue for each channel to receive messages in between marker sending and receiving in a channel.
- After receiving all the marker message print the snapshot of the system.
- Reset and initialize the queue once the snapshot is done.

- **Add 'Print Total money' features In the program:**

It becomes difficult to add each money token in each channel whenever the output comes so we have added a new features to print the total money of the system so that we don't have calculate manually.

- **Run and test:**

Again we have run the program for enough time and capture the output. Here we have tested the output for ten numbers of node.

5.0 EVALUATION METHODOLOGY & RESULTS

■ PHASE-I OUTPUT:

```
172.16.5.33>mpirun --hostfile my_host -np 10 moneyTransfer
1
Final balance of rank 4 is 16810
Final balance of rank 0 is 15951
Final balance of rank 7 is 3149
Final balance of rank 6 is 6111
Final balance of rank 9 is 16399
Final balance of rank 3 is 5700
Final balance of rank 8 is 4691
Final balance of rank 2 is 12013
Final balance of rank 1 is 16359
Final balance of rank 5 is 2817
Total Money in system = 100000
172.16.5.33>
```

Here, number of nodes in the distributed system is ten and by pressing '1' key in the key board we can see the output.

As we can see the total Money in System is 100,000 which is 10 times of 10,000 (initial money). Hence our result is correct and the test is successful.

■ PHASE-II OUTPUT:

Test 1:

```
172.16.5.33>mpirun -np 10 -pernode --hostfile my_host money_el
1
p0      171      {}      {14,16,}      {}      {30,}      {}      {}      {}      {}      {}      {}
p3      2419     {}      {}      {36,}      {}      {}      {}      {9,}      {26,}      {}      {}
p1      2332     {}      {}      {}      {}      {4,}      {}      {}      {23,}      {}      {18,}
p4      1880     {}      {}      {}      {}      {}      {}      {}      {}      {44,}      {}
p2      1192     {}      {17,}      {}      {}      {}      {}      {}      {}      {}      {46,}      {}
p6      512      {}      {}      {}      {}      {}      {}      {}      {10,}      {14,}      {18,}
p9      471      {}      {}      {44,}      {}      {11,}      {21,}      {}      {}      {}      {}
p7      87       {}      {}      {}      {}      {}      {}      {}      {}      {}      {}      {}
p8      383      {}      {}      {}      {}      {}      {}      {}      {22,}      {}      {}
p5      130      {}      {}      {}      {}      {}      {}      {}      {}      {}      {}

```

Test 2:

```
1
p0      1824     {}      {}      {}      {}      {}      {}      {}      {}      {}      {}
p1      502      {}      {}      {44,}      {25,34,}      {}      {}      {}      {}      {}      {}
p3      55       {}      {23,}      {11,}      {}      {30,}      {}      {23,}      {}      {}      {}
p4      889      {}      {}      {}      {}      {}      {30,}      {}      {18,}      {20,}      {24,6,}
p2      281      {}      {}      {}      {}      {20,}      {}      {}      {23,}      {44,}      {}
p6      865      {}      {}      {}      {}      {}      {}      {}      {}      {}      {42,}
p5      3976     {}      {}      {}      {}      {}      {}      {}      {41,}      {}      {}
p7      226      {}      {}      {}      {}      {}      {}      {}      {}      {}      {}
p9      583      {}      {10,}      {}      {}      {}      {}      {}      {}      {}      {}
p8      284      {}      {}      {}      {}      {}      {}      {}      {46,}      {}      {1,}

```

Here, we have again tested our chandy lamport algorithm on ten numbers of nodes. If we add the money in the channel and in the node it will give the same total of ten times of initial value of the nodes.

- TEST 3 (After adding 'print total money' features)

```
vikram@vikram-To-be-filled-by-0-E-M ~/Downloads/4thYear_project $ mpirun --hostfile my_host -np 6 CLTotalMoney
1
p1      62      {}      {}      {9,}      {}      {}      {15,}
p0      3551    {}      {16,}    {}      {}      {}      {9,}
p2      360     {}      {}      {}      {44,}    {}      {17,}
p4      56      {}      {}      {}      {}      {}      {}
p3      805     {}      {41,}    {}      {}      {}      {}
p5      962     {25,}    {28,}    {}      {}      {}      {}
Total money of the system=6000
1
```

- TEST 4

```
1
p2      51      {}      {14,}    {}      {}      {24,34,}    {13,}    {}
p1      1463    {}      {}      {}      {}      {}      {25,}
p3      2510    {}      {37,}    {}      {}      {}      {}
p4      328     {48,}    {}      {}      {17,}    {}      {}
p0      1266    {}      {41,}    {}      {}      {47,}    {}
p5      15      {}      {}      {29,33,} {}      {5,}      {}
Total money of the system=6000
^Cvikram@vikram-To-be-filled-by-0-E-M ~/Downloads/4thYear_project $
```

Here we have added the print total money features so that we don't have to calculate manually. Since the unavailability lab we have tested the program in our own computer with six virtual nodes and It has given the correct output.

6.0 CONCLUSION

So we can conclude from our project that Chandy lamport algorithm can be efficiently implemented in a distributed system to take a snapshot. And this snapshot can be used to check consistency of a distributed system.

There are many other uses of this algorithm. Some are:

- Garbage collection
- Deadlock Detection
- Termination Detection
- Distributed Check pointing

7.0 REFERENCES

- 1.0 <https://www.open-mpi.org/>
- 2.0 <https://www.geeksforgeeks.org/multithreading-c-2/>
- 3.0 <https://mpitutorial.com/tutorials/mpi-introduction/>
- 4.0 https://en.wikipedia.org/wiki/POSIX_Threads
- 5.0 <https://www.geeksforgeeks.org/thread-functions-in-c-c/>
- 6.0 <https://www.geeksforgeeks.org/chandy-lamports-global-state-recording-algorithm/>
- 7.0 https://en.wikipedia.org/wiki/Chandy%E2%80%93Lamport_algorithm

8.0 APPENDIX: SYSTEM DESCRIPTION

- Fedora v30 (64 bit OS, 4 GB RAM)
- OPEN MPI version 4.0.0
- GCC compiler should be configured.