Implementation of Chandy Lamport algorithm

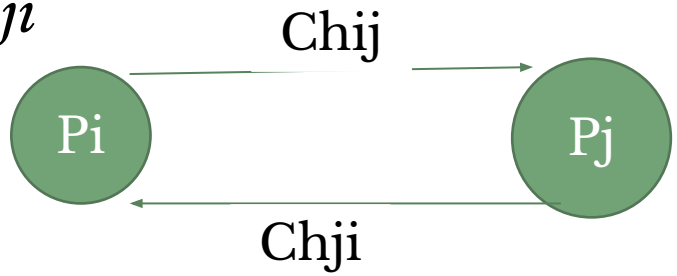
By
Kaushik Rajak(Roll No:001611001029)
Vikram Saha(Roll No: 001611001016)
Under the guidance of
Mr. Utpal Kumar Ray

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 traverse the distributed system across all communication channel and cause each process to record its own state. In the end, the state of entire system (Global state) is recorded. This algorithm does not interfere with normal execution of processes.

Chandy-Lamport algorithm

Initiator process *P0* records its state locally
2. Marker sending rule for process *Pi:*
-After *Pi* has recorded its state, for each outgoing channel *Chij*, *Pi* sends *one marker* message over *Chij* (before *Pi* sends any other message over *Chij*)
3. Marker receiving rule for process *Pi*:
-Process *Pi* on receipt of a marker over channel *Chji*
-If (*Pi* has not yet recorded its state)
--It Records *its process state* now;
--Records the state of *Chji* as *empty set;*
--Starts recording messages arriving over other incoming channels;
-else (*Pi* has already recorded its state)
--*Pi* records the state of *Chji* as the set of all messages it has received over *Chji* since *it* saved its state



PRE REQUISITES:

- OPEN MPI (Version 4.0.0)
- C programming language
- Linux Operating System (Fedora or Ubantu)
- POSIX Threads

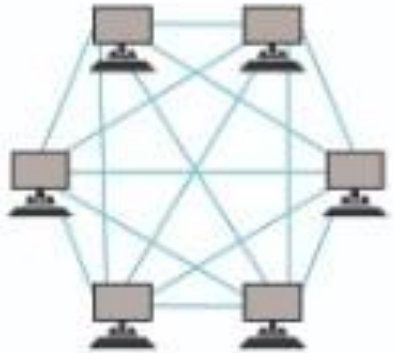
Some pre requisites concepts are explained in the upcoming slides.

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.
- Uses:

  --Creating Distributed environment.

  --Parallel computing.



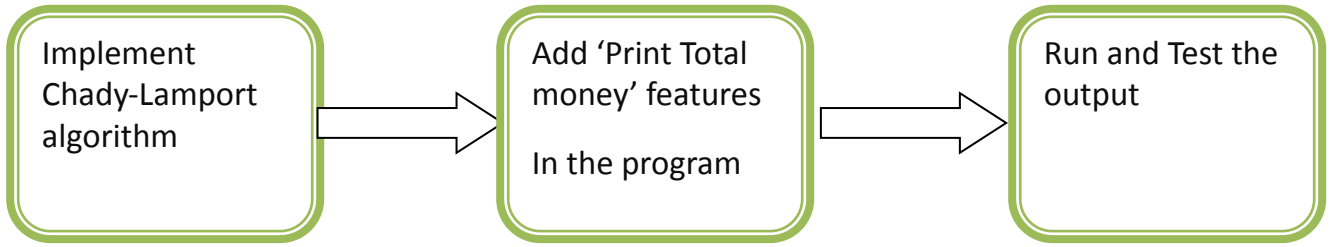
Some important functions in MPI:

- MPI_Comm_size (MPI_Comm communicator,int *size)
- MPI_Comm_rank(MPI_Comm Communicator,int *rank)
- MPI_Send(void* data, int count, MPI_Datatype datatype, int destination, int tag, MPI_Comm communicator)
- MPI_Recv(Void* data,int count,MPI_Datatype datatype,int source,int tag,MPI_Comm communicator,MPI_Status* status)
- MPI_Finalize()

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 remain 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.

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.

PHASE-II OUTPUT:

Test 1:

```
172.16.5.33>mpirun -np 10 -pernode --hostfile my_host money_e1
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-O-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
```

```
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-O-E-M ~/Downloads/4thYear_project $
```

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

REFERENCES:

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

Thank you.