**Amrita School of Engineering**

**Department of Computer Science and Engineering**

**19CSE312: Distributed Systems**

**Lab-Evaluation-1**

**Set -1**

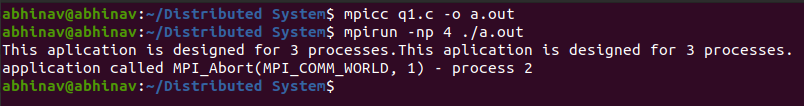
**Date: 04/02/2022** **Topic: MPI** **Time: 2hrs**

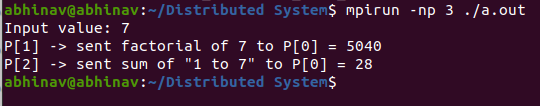
1. **Implement a MPI C program that**
2. **Computes factorial of a number in process 1**
3. **Generate sum of the series from 1 to n in process 2 given a common value of n sent by process 0. Display the results computed by each process and show the output in Process 0.**

**Code:**

#include <mpi.h>  
#include <stdio.h>  
#include <stdlib.h>  
  
#define ROOT 0  
#define FACT 1  
#define ADD 2  
  
int main(int argc, char\* argv[])  
{  
 int myrank, procount;  
 int new, commonval;  
  
 MPI\_Status status;  
 MPI\_Init(&argc, &argv);  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myrank);  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &procount);  
  
 int final;  
 if(procount != 3)  
 {  
 printf("This aplication is designed for 3 processes.\n");  
 MPI\_Abort(MPI\_COMM\_WORLD, EXIT\_FAILURE);  
 }  
 else if (myrank == ROOT){  
 printf("Input value: ");  
 scanf("%d", &commonval);   
 int msg = commonval;  
   
 for (new = 1; new < procount; new++)  
 {  
 if(new % 2 != ROOT)  
 MPI\_Send(&msg, 1, MPI\_INT, new, 1, MPI\_COMM\_WORLD);   
 else  
 MPI\_Send(&msg, 1, MPI\_INT, new, 2, MPI\_COMM\_WORLD);  
 }   
 for (new = 1; new < procount; new++)  
 {  
 if(new == 1)  
 {  
 MPI\_Recv(&final, 1, MPI\_INT, new, 1, MPI\_COMM\_WORLD, &status);  
 printf("P[%d] -> sent factorial of %d to P[%d] = %d\n", new, msg, myrank, final);  
 }  
 else  
 {  
 MPI\_Recv(&final, 1, MPI\_INT, new, 2, MPI\_COMM\_WORLD, &status);  
 printf("P[%d] -> sent sum of \"1 to %d\" to P[%d] = %d\n", new, msg, myrank, final);  
 }  
 }  
 }   
 else if (myrank % 2 != ROOT){  
 MPI\_Recv(&final, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, &status);   
 int out = 1;  
 for (new = 1; new <= final; new++){  
 out = out \* new;  
 }  
 MPI\_Send(&out, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD);  
 }   
 else{  
 MPI\_Recv(&final, 1, MPI\_INT, 0, 2, MPI\_COMM\_WORLD, &status);  
 int out = (final \* (final + 1)) / 2;  
 MPI\_Send(&out, 1, MPI\_INT, 0, 2, MPI\_COMM\_WORLD);  
 }  
 MPI\_Finalize();  
 return EXIT\_SUCCESS;  
}

**Output:**

****

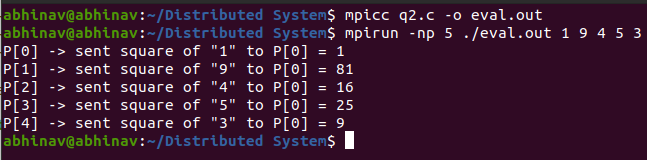
****

1. **Write a collective communication-based program in which the broadcaster initializes the vectors X and slave processes to compute the squares of their share of the vector. Collect the output from each process and print the output in Process 0.**

**Code:**

#include<stdio.h>  
#include<stdlib.h>  
#include<string.h>  
#include<mpi.h>  
  
#define ROOT 0  
  
int sqr(int x){  
 return x\*x;  
}  
int main(int argc, char \*\*argv)  
{  
  
 MPI\_Init(NULL, NULL);  
 int procount, myrank, value;  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &procount);  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myrank);  
 int x;  
 if(myrank == ROOT){  
 int vector[procount], final[procount];  
 for(int i=0; i<procount; i++)  
 vector[i] = atoi(argv[i+1]);  
 MPI\_Barrier(MPI\_COMM\_WORLD);  
 MPI\_Scatter(vector, 1, MPI\_INT, &x, 1, MPI\_INT, ROOT, MPI\_COMM\_WORLD);  
 value = sqr(x);  
 MPI\_Gather(&value, 1, MPI\_INT, final, 1, MPI\_INT, ROOT, MPI\_COMM\_WORLD);  
 for(int i=0; i<procount; i++)  
 printf("P[%d] -> sent square of \"%d\" to P[%d] = %d\n", i, vector[i], ROOT, final[i]);  
 }else{  
 MPI\_Barrier(MPI\_COMM\_WORLD);  
 MPI\_Scatter(NULL, 1, MPI\_INT, &x, 1, MPI\_INT, ROOT, MPI\_COMM\_WORLD);  
 value = sqr(x);  
 MPI\_Gather(&value, 1, MPI\_INT, NULL, 1, MPI\_INT, ROOT, MPI\_COMM\_WORLD);  
 }  
  
 MPI\_Finalize();  
 return EXIT\_SUCCESS;  
}

**Output:**

****