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**
   a. **Computes factorial of a number in process 1**
   b. **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:

```
abhinav@abhinav:~/Distributed System$ mpicc q1.c -o a.out
abhinav@abhinav:~/Distributed System$ mpirun -np 4 ./a.out
This aplication is designed for 3 processes.This aplication is designed for 3 processes.
application called MPI_Abort(MPI_COMM_WORLD, 1) - process 2
abhinav@abhinav:~/Distributed System$
```

```
abhinav@abhinav:~/Distributed System$ mpirun -np 3 ./a.out
Input value: 7
P[1] -> sent factorial of 7 to P[0] = 5040
P[2] -> sent sum of "1 to 7" to P[0] = 28
abhinav@abhinav:~/Distributed System$
```

2. **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:

```
abhinav@abhinav:~/Distributed System$ mpicc q2.c -o eval.out
abhinav@abhinav:~/Distributed System$ mpirun -np 5 ./eval.out 1 9 4 5 3
P[0] -> sent square of "1" to P[0] = 1
P[1] -> sent square of "9" to P[0] = 81
P[2] -> sent square of "4" to P[0] = 16
P[3] -> sent square of "5" to P[0] = 25
P[4] -> sent square of "3" to P[0] = 9
abhinav@abhinav:~/Distributed System$
```